Cohesive Subgraph Identification in Large Graphs

Mr. Kai Yao

This thesis is presented as part of the requirements for the conferral of the degree:

Doctor of Philosophy (PhD)

Supervisor:
Dr. Lijun Chang

The University of Sydney
School of Computer Science

2023

Declaration

I, *Mr. Kai Yao*, declare that this thesis is submitted in partial fulfilment of the requirements for the conferral of the degree *Doctor of Philosophy (PhD)*, from the University of Sydney, is wholly my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualifications at any other academic institution.

Signature:

Date: 20/1/2023

Acknowledgements

First of all, I would like to express my gratitude to my supervisor Dr. Lijun Chang for his continuous support and guidance during my PhD studies. He is professional, efficient, conscientious and patient. Whenever I encounter difficulties or have new ideas, Lijun is always there to help me, such as discussing the novelty and feasibility of the new ideas, inspiring me to think about the questions from a different viewpoint or sharing some relevant papers with me. Lijun also gives me invaluable advice in other aspects, including how to implement the algorithms efficiently in practice, how to make a good presentation, how to review papers and so on. All of these help me to acquire the necessary research skills and to be a better PhD student. I am fortunate in having Lijun as my supervisor during my PhD studies.

I would also like to express my gratitude to my father Mr. Hongwei Yao, my mother Ms. Qiuling Cai, and my wife Ms. Bei Zheng for their love and encouragement. During my PhD studies, the most difficult period of time would be the first half year of 2020. At that time, I was a first-year PhD student and felt anxious about my research work. What was worse, I could not come back to campus because of the COVID-19 pandemic. I worried that I could not complete my PhD successfully. My parents and wife gave me a lot of comfort and encouragement, which helped me to go through that tough time.

Abstract

Graph data is ubiquitous in real world applications, as the relationship among entities in the applications can be naturally captured by the graph model. Finding cohesive subgraphs is a fundamental problem in graph mining with diverse applications. Given the important roles of cohesive subgraphs, this thesis focuses on cohesive subgraph identification in large graphs.

Firstly, we study the size-bounded community search problem that aims to find a subgraph with the largest min-degree among all connected subgraphs that contain the query vertex q and have at least ℓ and at most h vertices, where q, ℓ, h are specified by the query. As the problem is NP-hard, we propose a branch-reduce-and-bound algorithm SC-BRB by developing nontrivial reducing techniques, upper bounding techniques, and branching techniques. Specifically, we develop three reduction rules, degree-based reduction, distance-based reduction, and inclusion-based reduction, to reduce the size of an instance before generating new branches/recursions. We design three upper bounds, degree-based upper bound, neighbor reconstruction-based upper bound, and degree classification-based upper bound, for pruning branches. We devise domination-based branching to reduce the number of branches. In addition, we also extend our techniques to enumerate all size-bounded communities efficiently.

Secondly, we formulate the notion of similar-biclique in bipartite graphs which is a special kind of biclique where all vertices from a designated side are similar to each other, and aim to enumerate all maximal similar-bicliques. The naive approach of first enumerating all maximal bicliques and then extracting all maximal similar-bicliques from them is inefficient, as enumerating maximal bicliques is time consuming. We propose a backtracking algorithm MSBE to directly enumerate maximal similar-bicliques, and power it by vertex reduction and optimization techniques. In addition, we design a novel index structure to speed up a time-critical operation of MSBE, as well as to speed up vertex reduction. Efficient index construction algorithms are developed. To handle dynamic graph updates, we propose algorithms and optimization techniques for maintaining our index. We also parallelize our index construction algorithms to exploit multiple CPU cores.

Thirdly, we consider balanced cliques in signed graphs — a clique is balanced if its vertex set can be partitioned into C_L and C_R such that all negative edges are between C_L and C_R — and study the problem of maximum balanced clique computation that aims to find the balanced clique C^* such that $\min\{|C^*_L|, |C^*_R|\} \geq \tau$ for a user-given threshold τ and $|C^*|$ is the largest possible. Our main idea is a novel graph reduction that transforms a balanced clique problem over a signed graph G to problems over small subgraphs of G. Specifically, for each vertex u in G, we extract the subgraph

G_u of G induced by $V_L \cup V_R$; V_L is u and u's positive neighbors while V_R is u's negative neighbors. Then, we remove from G_u all positive edges between V_L and V_R and all negative edges between vertices of the same set; denote the resulting graph of discarding edge signs as g_u. We show that all balanced cliques containing u in G can be found by processing g_u. Due to the small size and no edge signs, large cliques containing u in g_u can be efficiently identified. Furthermore, we extend our techniques to the large balanced clique enumeration problem that aims to find maximal balanced cliques whose size is near the largest, to the polarization factor problem which aims to find the largest τ such that there is a balanced clique C with $\min\{|C_L|, |C_R|\} \geq \tau$, and to the generalized maximum balanced clique problem that reports a maximum balanced clique for each $\tau \geq 0$.

Publications During Candidature

- **Kai Yao**, Lijun Chang. "Efficient Size-Bounded Community Search over Large Networks." In Proceedings of the VLDB Endowment, 2021, 14(8): 1441-1453.
- **Kai Yao**, Lijun Chang, Jeffrey Xu Yu. "Identifying Similar-Bicliques in Bipartite Graphs." In Proceedings of the VLDB Endowment, 2022, 15(11): 3085-3097.
- **Kai Yao**, Lijun Chang, Lu Qin. "Computing Maximum Structural Balanced Cliques in Signed Graphs." In 2022 IEEE 38th International Conference on Data Engineering (ICDE). IEEE, 2022: 1004-1016.
- Lijung Chang, Xing Feng, **Kai Yao**, Lu Qin, Wenjie Zhang. "Accelerating Graph Similarity Search via Efficient GED Computation." IEEE Transactions on Knowledge and Data Engineering, 2022.
- Tran Ba Trung, Lijun Chang, Nguyen Tien Long, **Kai Yao**, Huynh Thi Thanh Binh. "Verification-Free Approaches to Efficient Locally Densest Subgraph Discovery." In 2023 IEEE 39th International Conference on Data Engineering (ICDE).

Authorship Attribution Statement

- Chapter 3 of this thesis is published as (Yao and Chang, 2021). I formulated the problems, designed the algorithms, conducted the experiments and wrote the drafts of the paper.
- Chapter 4 of this thesis is published as (Yao et al., 2022b). I formulated the problems, designed the algorithms, conducted the experiments and wrote the drafts of the paper.
- Chapter 5 of this thesis is published as (Yao et al., 2022a). I formulated the problems, designed the algorithms, conducted the experiments and wrote the drafts of the paper.
- In all of the presented research in this thesis, Dr. Lijun Chang, as my lead supervisor, has provided technical guidance for formulating the problems, refinement of ideas as well as reviewing and polishing the presentation.

I declare that the authorship attribution statements above are true and correct.

Student Name: Kai Yao

Signature:

Date: 20/1/2023

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are true and correct.

Supervisor Name: Lijun Chang

Signature:

Date: 20/1/2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graph is a fundamental type of data structure to represent entities and their relationships in the real-world. Given a graph $G = (V, E)$, vertices in V represent the entities of interest and edges in E represent the relationships between the entities. Cohesive subgraph identification is an important problem in graph mining with diverse applications such as recommendation (Bedi and Sharma, 2016), anomaly detection in financial networks (Ahmed et al., 2016) and protein complexes detection in protein-to-protein networks (Suratanee et al., 2014). Generally speaking, cohesive subgraph identification can be roughly classified into two categories. The first one is query-dependent that aims to find cohesive subgraphs that contain the pre-specified query vertex (Fang et al., 2020a), and another one is non-query-dependent that aims to find all the cohesive subgraphs that satisfy particular criteria (Lee et al., 2010). To better reflect the relationships among entities, cohesive subgraph has been formulated and extensively studied on different types of graphs such as ordinary graphs, bipartite graphs, signed graphs, attributed graphs and so on. For example, suspicious groups and illegal financial activities in customer-product networks can be modeled as cohesive subgraphs in bipartite graphs (Yu et al., 2021). Polarized communities on modern social-media platforms (*e.g.*, Reddit) can be modeled as cohesive subgraphs in signed graphs (Bonchi et al., 2019). With all the applications above, this thesis focuses on the cohesive subgraph identification problem. We study the size-bounded community search problem in ordinary graphs, which belongs to the query-dependent cohesive subgraph identification problem. Besides, we also study two non-query-dependent cohesive subgraph identification problems. One is similar-biclique computation in bipartite graphs and another one is balanced clique computation in signed graphs.

1.1 Size-Bounded Community Search

Graph is widely used to model the relationships among entities in the real-world. Graph-based data analytics, as a result, has become increasingly popular, among which cohesive subgraph computation is a fundamental problem (Chang and Qin, 2019). Much of the recent focus of cohesive subgraph computation is on query-dependent cohesive subgraph search, also known as community search, which aims to find cohesive subgraphs that contain user-specified query vertices and possibly

satisfy some other constraints (Fang et al., 2020a; Huang et al., 2019). Community search has a wide range of applications, such as event organization (She et al., 2016, 2017), social marketing (Manchanda et al., 2015), task scheduling (Sinnen and Sousa, 2005), and social network analysis (Knoke and Yang, 2019; Scott, 1988).

The basic setting is as follows: given a large graph $G = (V, E)$ and a query vertex $q \in V$, community search aims to find a subgraph of G that contains q and is most cohesive (*e.g.*, densest). One of the most popular cohesiveness measures adopted in the literature is the minimum degree (Bi et al., 2018; Cui et al., 2014; Sozio and Gionis, 2010), which is also the one we adopt in this thesis. That is, it aims to find the one with the largest min-degree among all subgraphs of G that contain q. As there could exist many subgraphs having the same min-degree, the existing studies return either a maximal one (Bi et al., 2018) or an arbitrary one (Cui et al., 2014; Sozio and Gionis, 2010) among all subgraphs with the largest min-degree. The maximal subgraph with the largest min-degree can be computed in linear time (Sozio and Gionis, 2010), by iteratively peeling the vertex with the smallest degree; the optimal result is among these n (*i.e.*, $|V|$) subgraphs. However, the result could be extremely large which may overwhelm end-users (Chang and Qin, 2019).

It is observed in (Sozio and Gionis, 2010) that limiting the size of the returned community, to accommodate resource limitations, is natural and interesting from application point of view: for example, organize a hiking trip with up to 15 attendees, assemble a team of up to 10 workers for a project (Ma et al., 2019). Motivated by this, we investigate the general size-bounded community search (SCS) problem. Formally speaking, given a large graph G, a query vertex q and a size constraint $[\ell, h]$, the SCS problem aims to find a subgraph with the largest min-degree among all connected subgraphs of G that contain q and have at least ℓ and at most h vertices. As the problem is NP-hard, we propose a branch-reduce-and-bound algorithm SC-BRB by developing nontrivial reducing techniques, upper bounding techniques, and branching techniques. Specifically, we develop three reduction rules, degree-based reduction, distance-based reduction, and inclusion-based reduction, to reduce the size of an instance before generating new branches/recursions. We also design three upper bounds, degree-based upper bound, neighbor reconstruction-based upper bound, and degree classification-based upper bound, for pruning branches. In addition, we also devise domination-based branching to reduce the number of branches. Finally, we extend our techniques to enumerate all size-bounded communities efficiently.

1.2 Similar-Biclique Identification

Bipartite graphs have been widely used in real-world applications to model relationships between entities of different types, such as customer-product networks (Wang et al., 2006), author-paper networks (Ley, 2002) and user-event networks (EL BACHA and Zin, 2018). A bipartite graph is denoted by $G = (V_L, V_R, E)$, where the vertex set is partitioned into two disjoint subsets V_L and V_R which are referred to as the L-side and R-side vertices of the bipartite graph, respectively; each edge $e \in E$ can only connect vertices from different sides. Finding dense subgraphs in a bipartite graph is of great significance and encompasses many applications, such as community

detection (Abidi et al., 2021; Lehmann et al., 2008), anomaly detection (Gangireddy et al., 2020; Sarıyüce and Pinar, 2018), and group recommendation (Lyu et al., 2020; Su and Khoshgoftaar, 2009).

One classic notion of dense bipartite subgraph is biclique (Peeters, 2003), which requires every pair of vertices from different sides of the subgraph to be connected by an edge. In the literature, many algorithms have been proposed to enumerate all maximal bicliques (Abidi et al., 2021; Alexe et al., 2004; Eppstein, 1994; Li et al., 2007; Liu et al., 2006; Uno et al., 2004; Zhang et al., 2014) and to identify a biclique of the maximum size (Lyu et al., 2020). However, the biclique model has a fundamental limitation: vertices in a biclique are not necessarily similar to each other, despite that they share a set of common neighbors (i.e., vertices on the other side of the biclique).

Motivated by this, we formulate the notion of similar-biclique by requiring all vertices from one side of the biclique to be similar to each other. Formally speaking, given a similarity threshold $0 < \varepsilon \leq 1$ and a size constraint $\tau \geq 0$, a vertex subset $C \subseteq V_L \cup V_R$ in a bipartite graph $G = (V_L, V_R, E)$ is a similar-biclique if (1) C is a biclique (i.e., $C_L \times C_R \subseteq E$), (2) all vertices of C_L are similar to each other (i.e., $\mathsf{sim}(u, v) \geq \varepsilon, \forall u, v \in C_L$), and (3) C satisfies the size constraint (i.e., $|C_L| \geq \tau$ and $|C_R| \geq \tau$). Here, C_L denotes $C \cap V_L$ and C_R denotes $C \cap V_R$; $\mathsf{sim}(u, v)$ measures the structural similarity between u and v, which is computed based on their neighbors $N(u)$ and $N(v)$ and will be formally defined in Section 4.2; the size constraint τ is introduced to avoid generating too small or too skewed results. We propose a backtracking algorithm MSBE to directly enumerate maximal similar-bicliques, and power it by vertex reduction and optimization techniques. In addition, we design a novel index structure to speed up a time-critical operation of MSBE, as well as to speed up vertex reduction. Efficient index construction algorithms are developed. To handle dynamic graph updates, we also propose algorithms and optimization techniques for maintaining our index. Finally, we parallelize our index construction algorithms to exploit multiple CPU cores.

1.3 Structural Balanced Clique Identification

Signed graphs enhance the representation capability of traditional graphs, by capturing the polarity of relationships between entities/vertices through positive and negative edge signs (Tang et al., 2016b). For example, signed graphs capture the friend-foe relationship in social networks (Easley and Kleinberg, 2010), support-dissent opinions in opinion networks (Kunegis et al., 2009) and activation-inhibition in protein-protein interaction networks (Ou-Yang et al., 2015). One prominent and fundamental theory in signed graph analysis is the structural balance theory (Harary et al., 1953), which states that a signed (sub)graph is structural balanced if its vertices can be partitioned into two sets such that all edges inside each partition have positive signs and all cross-partition edges have negative signs. Many interesting problems, such as community detection (Chu et al., 2016; Ordozgoiti et al., 2020), link prediction (Leskovec et al., 2010; Ye et al., 2013) and recommendation systems (Chen et al., 2013; Tang et al., 2016a), have been formulated and studied for signed graphs based on the structural balance theory.

Recently, the problem of enumerating all maximal structural balanced cliques in a signed graph is formulated and studied in (Chen et al., 2020). A vertex set C is a structural balanced clique if (1) it is a clique (i.e., every pair of its vertices is connected by an edge), and (2) it is structural balanced according to the structural balance theory (i.e., C can be partitioned into two sets C_L and C_R such that all negative edges are between C_L and C_R). For presentation simplicity, we refer to structural balanced cliques as balanced cliques. However, signed graphs may have an enormous number of maximal balanced cliques, of varying sizes. Enumerating all of them may overwhelm end-users. To remedy this, a threshold τ is adopted in (Chen et al., 2020) such that only maximal balanced cliques C satisfying $|C_L| \geq \tau$ and $|C_R| \geq \tau$ are enumerated. Nevertheless, the number of such cliques could still be large, and the efficiency of the algorithms in (Chen et al., 2020) is not satisfactory.

Motivated by this, we investigate both the maximum balanced clique computation problem and the large balanced clique enumeration problem. Given a signed graph $G = (V, E^+, E^-)$ and a threshold τ, the maximum balanced clique computation problem aims to find the largest balanced clique C^* in G that satisfies $|C_L^*| \geq \tau$ and $|C_R^*| \geq \tau$. Let $\omega_\tau(G)$ be the size of C^*, i.e., $\omega_\tau(G) = |C^*|$. Given G and two thresholds τ and α, the large balanced clique enumeration problem aims to enumerate all maximal balanced cliques $C \subseteq V$ that satisfy $|C_L| \geq \tau$, $|C_R| \geq \tau$ and $|C| \geq \omega_\tau(G) - \alpha$. We propose branch and bound algorithms to solve the above two problems efficiently. Our main technique is a novel graph reduction that transforms a balanced clique problem over a signed graph G to problems over small subgraphs of G.

Both our maximum balanced clique computation problem and the maximal balanced clique enumeration problem studied in (Chen et al., 2020) require a user-given threshold τ. However, it is unclear how to choose the appropriate τ for an application. Choosing a too large τ may lead to no result, while a too small τ may lead to skewed results as well as an enormous number of results for the enumeration problem of (Chen et al., 2020). We provide two alternative ways to resolve this issue. Firstly, we investigate the polarization factor problem, which computes the largest τ^* such that G has a balanced clique C satisfying $|C_L| \geq \tau^*$ and $|C_R| \geq \tau^*$. We call this τ^* the polarization factor of G, denoted $\beta(G)$. It is immediate that there is no balanced clique for $\tau > \beta(G)$. Secondly, we investigate the generalized maximum balanced clique problem, which computes a maximum balanced clique for every $\tau \geq 0$.

1.4 Our Contributions

Our main contributions are summarized as follows.

- *Formulation of size-bounded community and efficient algorithms for size-bounded community search (Chapter 3).* We formulate a new community model by putting size constraints on the extracted community, *i.e.*, size-bounded community. However, detecting the size-bounded community is NP-hard. We propose a branch and bound algorithm to solve this problem. We also design novel reducing techniques, upper bounding rules and branching rules to facili-

tate the computation. Experimental studies on large real graphs demonstrate the efficiency and effectiveness of our algorithms.

- *Formulation of similar-biclique model and efficient algorithms for the maximal similar-biclique enumeration (Chapter 4).* We formulate a novel cohesive subgraph model in bipartite graphs called similar-biclique, which is a special kind of biclique where all vertices from a designated side are similar to each other. However, detecting all maximal similar-bicliques is #P-complete. We propose a backtracking algorithm to solve this problem and improve the efficiency by designing a novel index structure. Experimental studies on large real bipartite graphs demonstrate the efficiency of our algorithms and the effectiveness of our similar-biclique model.

- *Efficient algorithms for the maximum balanced clique computation problem and its variants (Chapter 5).* We propose an efficient algorithm for the maximum balanced clique computation in signed graphs. A novel graph reduction technique is proposed to facilitate the computation. We also extend our techniques to the large balanced clique enumeration problem, the polarization factor problem and the generalized maximum balanced clique problem. Experimental studies on large real signed graphs demonstrate the efficiency and effectiveness of our techniques.

1.5 Organization

Chapter 2 reviews the related works. Chapter 3 studies the size-bounded community search problem in ordinary graphs. Chapter 4 studies the similar-biclique identification problem in bipartite graphs. Chapter 5 studies the structural balanced clique identification problem in signed graphs. Chapter 6 concludes the thesis.

Chapter 2

Literature Review

In this chapter, we first review the cohesive subgraph computation in ordinary graphs in Section 2.1. Secondly, we review the cohesive subgraph computation in bipartite graphs in Section 2.2. Thirdly, we review the cohesive subgraph computation in signed graphs in Section 2.3. Lastly, we review the community search in Section 2.4.

2.1 Cohesive Subgraph Computation in Ordinary Graphs

Cohesive subgraph computation is a fundamental research topic in graph mining, which aims to find subgraphs that are densely linked internally in the graph. Clique is a classical cohesive subgraph model that requires each vertex is adjacent to every other vertex. The maximal clique enumeration problem (Bron and Kerbosch, 1973; Chang et al., 2013a; Eppstein et al., 2013; Tomita et al., 2006) and the maximum clique computation problem (Chang, 2019; Li et al., 2013, 2017; Tomita, 2017; Tomita et al., 2010) have been extensively studied in the literature. However, the clique model may be overly restrictive for many application scenarios. Many other relaxed forms of clique are proposed. k-plex (Seidman and Foster, 1978) relaxes the degree constraint of the clique by allowing each vertex to miss up to k vertices. γ-quasi-clique (Abello et al., 2002) relaxes the edge density of the clique by allowing the edge density to be γ. n-clique (Mokken et al., 1979) relaxes the distance constraint between any two vertices from 1 to n. n-clan (Mokken et al., 1979) is an n-clique in which the diameter should be no larger than n. n-club (Mokken et al., 1979) is a subgraph whose diameter should be no larger than n without n-clique restriction. However, cohesive subgraph computation based on these definitions usually leads to NP-hard problems.

Other cohesive subgraph models are also proposed in the literature, such as densest subgraph (Goldberg, 1984), k-core (Batagelj and Zaversnik, 2003), k-truss (Cohen, 2008) and k-edge connected component (Chang et al., 2013b; Zhou et al., 2012). Densest subgraph is the subgraph with the largest average degree. k-core is the maximal subgraph whose minimum degree is at least k. k-truss is the maximal subgraph in which each edge is involved in at least k triangles within the subgraph. k-edge connect component is the subgraph which is still connected after removing

any k edges.

2.2 Cohesive Subgraph Computation in Bipartite Graphs

As a counterpart of the clique model in unipartite graphs, biclique model is formulated to capture the cohesive subgraphs in bipartite graphs (Hochbaum, 1998). A lot of literature investigated the problem of enumerating maximal bicliques (Abidi et al., 2021; Li et al., 2005, 2007; Sanderson et al., 2003; Uno et al., 2004; Zaki and Hsiao, 2002; Zhang et al., 2014). The existing studies can be classified into two categories, depending on whether the input graph is bipartite or not. When the input graph is bipartite, the existing algorithms (Abidi et al., 2021; Sanderson et al., 2003; Zhang et al., 2014) generally enumerate subsets of vertices from one side, and then the intersection of their neighbors form the other side of the biclique. Besides, frequent item-set mining techniques have also been utilized to enumerate maximal bicliques (Li et al., 2005, 2007; Uno et al., 2004; Zaki and Hsiao, 2002), as these two problems are highly related to each other. The state-of-the-art algorithm for maximal biclique enumeration over bipartite graphs is ooMBEA proposed in (Chen et al., 2022), which is a backtracking algorithm equipped with a novel batch-pivots technique. There are also studies that aim at enumerating all maximal (non-induced) bicliques from a general graph, i.e., the input graph is not bipartite. For example, it is studied from a theoretical viewpoint in (Eppstein, 1994), consensus algorithms are proposed in (Alexe et al., 2004), and a divide-and-conquer algorithm is proposed in (Liu et al., 2006). However, these algorithms are generally slower than the algorithms that specifically handle bipartite graphs. The maximum biclique problem (i.e., the biclique with the largest number of edges) and its variants (i.e., the maximum vertex biclique and the maximum balanced biclique) have also been studied in the literature (Chen et al., 2021a; Dawande et al., 2001; Lyu et al., 2020; Shaham et al., 2016; Shahinpour et al., 2017; Zhou et al., 2018).

Besides biclique, other models have also been proposed for dense bipartite subgraph identification, such as quasi-biclique (Liu et al., 2008), k-biplex (Yu et al., 2021), (α, β)-core (Kumar et al., 1999), k-bitruss (Zou, 2016), and k-wing (Sarıyüce and Pinar, 2018). Quasi-biclique and k-biplex relax the biclique model by allowing each vertex in one side of the result to miss a certain number of neighbors from the other side. On the other hand, (α, β)-core requires each vertex from one side to be connected to a certain number of vertices from the other side, and k-bitruss and k-wing require each edge to be involved in a certain number of $(2, 2)$-bicliques.

2.3 Cohesive Subgraph Computation in Signed Graphs

Signed graph was firstly studied by Harary et al. (1953), where the notion of structural balanced is introduced. The problem of finding the largest (in terms of vertex number) vertex-induced subgraph that is structural balanced, known as the

maximum balanced subgraph problem, is studied in (Figueiredo and Frota, 2014; Ordozgoiti et al., 2020). The problem is NP-hard. A branch-and-cut exact algorithm, which only works for graphs with up-to a few thousand vertices, is proposed in (Figueiredo and Frota, 2014). Heuristic algorithms without any guarantee on the solution optimality are investigated in (Figueiredo and Frota, 2014; Ordozgoiti et al., 2020). Hao et al. (2014) introduced the notion of k-balanced trusted clique for signed graphs, which is a clique with k vertices such that all its edges are positive edges. This is essentially the same problem as the traditional k-clique problem over unsigned graphs, i.e., by removing all negative edges. The notion of (α, k)-clique is defined in (Li et al., 2018a) for signed graphs, which is a clique such that each vertex has at most k negative neighbors and at least αk positive neighbors in the clique. However, it does not require the extracted clique to be structural balanced. Recently, the problem of polarized community detection is studied in (Bonchi et al., 2019; Chu et al., 2016; Ordozgoiti et al., 2020), which aims to find multiple communities (i.e., subgraphs) where within communities there are mostly positive edges while across communities there are mostly negative edges. Chen et al. (2020) formulate the notion of (structural) balanced clique, which is a complete subgraph that can be partitioned into two sets such that all negative edges are between two sets. Motivated by the work in (Chen et al., 2020), Sun et al. (2022) study the problem of maximal balanced signed biclique enumeration in signed bipartite graphs.

2.4 Community Search

The community search problem was first proposed by Sozio and Gionis (2010). The basic setting is as follows: given a large graph $G = (V, E)$ and a query vertex $q \in V$, community search aims to find a connected subgraph of G that contains q and is cohesive. The cohesiveness is often evaluated by the classical cohesive subgraph metrics such as k-core and k-truss. In (Sozio and Gionis, 2010), k-core is used to evaluate the community cohesiveness. They design a global-search based algorithm which iteratively removes the vertex with the minimum degree. Later, Cui et al. (2014) improve the search efficiency by designing a local-search based algorithm, which incrementally expands the community from the query vertex. Many community search problems are also formulated based on k-truss. For example, Huang et al. (2014) formulate the concept of k-truss community in which each edge is involved in at least k triangles and every two edges are triangle connected. To avoid the free-rider-effect, Huang et al. (2015) formulate the closest truss community, which aims to detect a connected k-truss with the largest k that contains query vertices, and has the minimum diameter among such subgraphs. Other cohesive subgraph metrics are also used in community search problem. For example, clique-based community search is studied in (Cui et al., 2013; Yuan et al., 2017), k-edge-connected component based community search is studied in (Chang et al., 2015), modularity-based community search is studied in (Clauset, 2005; Kim et al., 2022).

Besides ordinary graphs, community search problem has also been studied on other types of graphs. Fang et al. (2017) and Chen et al. (2018) studied the spatial-aware community search problem on spatial graph, which aims to find the subgraph whose vertices are structurally cohesive and spatially close. Fang et al. (2016) and

Huang and Lakshmanan (2017) studied the community search problem on attributed graph, *i.e.*, vertices are associated with textual descriptions. The general target is to find the cohesive subgraph whose vertices have similar textual descriptions. Community search problem on directed graphs (Chen et al., 2021b; Fang et al., 2018; Liu et al., 2020), weighted graphs (Bi et al., 2018; Li et al., 2015), temporal graphs (Li et al., 2022, 2018b) and heterogeneous information graphs (Fang et al., 2020b) has also been extensively studied. For more details, please see the survey (Fang et al., 2020a).

Chapter 3

Size-Bounded Community Search

In this chapter, we study the size-bounded community search problem in large graphs. The work is published in (Yao and Chang, 2021). This chapter is organized as follows. Section 3.1 provides the introduction to this work. Section 3.2 provides the preliminaries and the problem statement. Section 3.3 introduces a baseline algorithm. Section 3.4 introduces our branch-reduce-and-bound algorithm. Section 3.5 presents our heuristic algorithm. Section 3.6 studies the size-bounded community enumeration problem. Experimental results are reported in Section 3.7. Section 3.8 concludes the chapter.

3.1 Introduction

Graph data is ubiquitous in real world applications, as the relationship among entities in the applications can be naturally captured by the graph model. Graph-based data analytics, as a result, has become increasingly popular, among which cohesive subgraph computation is a fundamental problem (Chang and Qin, 2019). Many of the recent focus of cohesive subgraph computation is on cohesive subgraph search, also known as *community search*, which aims to find cohesive subgraphs that contain user-specified query vertices and possibly satisfy some other constraints (Fang et al., 2020a; Huang et al., 2019). Community search has a wide range of applications, such as event organization (She et al., 2016, 2017), social marketing (Manchanda et al., 2015), task scheduling (Sinnen and Sousa, 2005), and social network analysis (Knoke and Yang, 2019; Scott, 1988).

The basic setting is as follows: given a large graph $G = (V, E)$ and a query vertex $q \in V$, community search aims to find a subgraph of G that contains q and is most cohesive (*e.g.*, densest). One of the most popular cohesiveness measures adopted in the literature is the *minimum degree* (Bi et al., 2018; Cui et al., 2014; Sozio and Gionis, 2010), which is also the one we adopt in this chapter. That is, it aims to find the one with the largest min-degree among all subgraphs of G that contain q. As there could exist many subgraphs having the same min-degree, the existing studies return either a maximal one (Bi et al., 2018) or an arbitrary one (Cui et al., 2014; Sozio and Gionis, 2010) among all subgraphs with the largest min-degree. The maximal subgraph with the largest min-degree can be computed in linear time (Sozio and Gionis, 2010), by iteratively peeling the vertex with the smallest degree; the

optimal result is among these n (*i.e.*, $|V|$) subgraphs. However, the result could be extremely large which may overwhelm end-users (Chang and Qin, 2019).

It is observed in (Sozio and Gionis, 2010) that limiting the size of the returned community, to accommodate resource limitations, is natural and interesting from application point of view: for example, organize a hiking trip with up to 15 attendees, assemble a team of up to 10 workers for a project (Ma et al., 2019). Motivated by this, the problem of community search with size restriction is formulated and studied in (Sozio and Gionis, 2010), which restricts the consideration to subgraphs with at most h vertices for a user-given h. The problem with size restriction becomes NP-hard, and two heuristic algorithms, GreedyD and GreedyF, are proposed in (Sozio and Gionis, 2010). Both algorithms involve solving the problem without size restriction (*i.e.*, compute the maximal subgraph with the largest min-degree) on subgraphs of G. Specifically, GreedyD iteratively solves the problem without size restriction on the subgraphs of G induced by $V_{\leq d}$ for different d values, where $V_{\leq d}$ is the set of vertices that are within d hops from q in G. It returns the solution of $V_{\leq d^*}$ where d^* is the largest d such that the solution size of $V_{\leq d}$ is at most h; if there is no such d, then it returns the solution of $V_{\leq 1}$. For time efficiency consideration, GreedyF simply extracts the set $S \subseteq V$ of h vertices that are closest to q in G, and then returns the solution of the problem without size restriction on the subgraph of G induced by S.[1] Both algorithms have no guarantee on the minimum degree of the returned community compared with the optimal one (*i.e.*, largest min-degree). We observe in our experiments that the communities returned by GreedyD and GreedyF are far from being optimal. We also observe that GreedyD usually returns a community with larger min-degree than GreedyF, but the size of the community returned by GreedyD may exceed h.

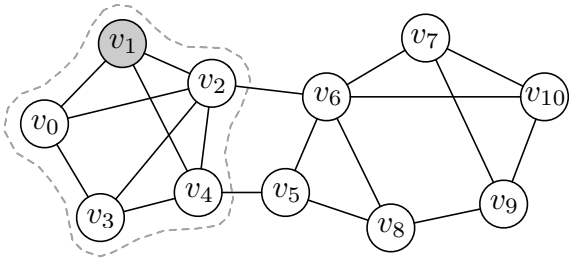


Figure 3.1: A toy graph

In this chapter, we aim to develop *exact* algorithms for the general *size-bounded community search* (SCS) problem, which in addition to the size upper bound h, also imposes a size lower bound ℓ. This is because some applications may also require the size of the identified community to be not too small. For example, consider the toy social network in Figure 3.1 of an event organization platform (*e.g.*, Meetup). Suppose user v_1 is planning to organize a group trip, and (s)he is deciding whom to invite. There are several considerations. Firstly, to enjoy a friendly atmosphere during the trip, it is desirable that every attendee has as many friends in the group as possible. Secondly, due to the limitation of accommodation, there can be at most

[1]Note that our descriptions here for GreedyD and GreedyF are simplified for the case that there is only one query vertex.

eight attendees. Thirdly, to be qualified for an air-ticket discount, the group should not be smaller than four. Thus, v_1 may issue an SCS query with size constraint $\ell = 4$ and $h = 8$, and find that $\{v_0, v_1, v_2, v_3, v_4\}$, which has minimum degree 3, is the best group. Note that, the existing approaches without size constraint may recommend the entire graph which also has minimum degree 3; however, this exceeds the accommodation capacity.

Formally speaking, given a large graph G, a query vertex q and a size constraint $[\ell, h]$, the SCS problem aims to find a subgraph with the largest min-degree among all connected subgraphs of G that contain q and have at least ℓ and at most h vertices. As expected, the SCS problem is NP-hard. A straightforward approach is to enumerate all subgraphs of G whose sizes are between ℓ and h and then identify the one with the largest min-degree. However, the time complexity would be $\Theta(n^h)$ which is too high to be practical. Note that, we cannot exploit the apriori-based pruning which has been demonstrated to be very successful for reducing the search space of frequent subgraph mining that also enumerates subgraphs of size up to a threshold (Inokuchi et al., 2000). This is because the property of minimum degree is not *hereditary*; that is, a graph can have subgraphs with smaller min-degree than itself, and may also have subgraphs with larger min-degree than itself.

In order to efficiently solve the SCS problem for large real graphs, we propose a branch-reduce-and-bound algorithm SC-BRB by developing nontrivial reducing techniques, upper bounding techniques, and branching techniques. Specifically, we develop three reduction rules, degree-based reduction, distance-based reduction, and inclusion-based reduction, to reduce the size of an instance before generating new branches/recursions. We also design three upper bounds, degree-based upper bound, neighbor reconstruction-based upper bound, and degree classification-based upper bound, for pruning branches. In addition, we also devise domination-based branching to reduce the number of branches. We rigorously proved the correctness of our techniques. For the SCS problem, we observe that the size-bounded community is not unique. Identifying all such communities can provide users with more choices. Motivated by this, we also extend our techniques to enumerate all size-bounded communities.

Contributions. Our main contributions are as follows.

- To the best of our knowledge, we are the first to study the general size-bounded community search (SCS) problem that has both a size lower bound and a size upper bound.
- We propose a branch-reduce-and-bound algorithm SC-BRB for solving the SCS problem exactly over large real graphs.
- We develop nontrivial reducing techniques, upper bounding techniques, and branching techniques for the SCS problem.
- We extend our techniques to enumerate all size-bounded communities.
- We conduct extensive experiments on large real graphs to demonstrate the effectiveness and efficiency of our algorihtms.

3.2 Preliminaries

We focus on a large undirected and unweighted graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. We denote the number of vertices and the number of edges in G by n and m, respectively. The set of neighbors of u in G is $N_G(u) = \{v \in V \mid (u, v) \in E\}$, and the degree of u in G is $d_G(u) = |N_G(u)|$. When the context is clear, $N_G(u)$ and $d_G(u)$ are simplified to $N(u)$ and $d(u)$, respectively. Given a vertex subset S of G, we use $G[S]$ to denote the subgraph of G induced by S, *i.e.*, $G[S] = (S, \{(u, v) \in E \mid u \in S, v \in S\})$. Given an arbitrary graph g, we use $V(g)$ and $E(g)$ to denote its set of vertices and its set of edges, respectively.

We measure the cohesiveness of a subgraph by its *minimum degree*. Minimum degree is a widely adopted cohesiveness measure in the literature (Bi et al., 2018; Cui et al., 2014; Li et al., 2019; Sozio and Gionis, 2010), due to its simplicity and easy computability. A related concept is *k-core*. Given a graph G and an integer k, the k-core of G is the *maximal* subgraph g of G such that $d_{\min}(g) \geq k$, where $d_{\min}(g)$ denotes the minimum degree of g. Note that, the k-core of G is unique for any $k \geq 0$, and is a vertex induced subgraph of G. The *core number* of a vertex v in G, denoted $\mathsf{cn}(v)$, is defined as the largest k such that the k-core of G contains v. It is well-known that computing the core number for all vertices in a graph, called the *core decomposition* problem, can be achieved in linear time (Batagelj and Zaversnik, 2003; Seidman, 1983).

Problem 1 (Size-Bounded Community Search)**.** Given a graph $G = (V, E)$, a query vertex $q \in V$, and a size constraint $[\ell, h]$, the Size-Bounded Community Search (SCS) problem aims to find a subgraph H of G such that it satisfies all of the three conditions:

1) Connected: H is connected and contains q;

2) Size Bounded: H satisfies the size constraint, *i.e.*, $\ell \leq |V(H)| \leq h$;

3) Cohesive: The minimum degree of H is maximized among all subgraphs of G satisfying the above two conditions.

An SCS query consists of a query vertex q, size lower bound ℓ, and size upper bound h. Note that, the aim of SCS query is not to recover the ground-truth communities, but to find densely-connected subgraphs that are of a certain size (*i.e.*, between ℓ and h). The parameters ℓ and h should be specified based on the resource limitations of the applications as illustrated in Introduction. For presentation simplicity, we assume there is only a single query vertex; nevertheless, our techniques can be extended to handle multiple query vertices. We call a subgraph of G that satisfies the first two conditions (*i.e.*, connected, and size bounded) a *feasible community* (or feasible solution), and a subgraph satisfying all the three conditions an *optimal community* (or optimal solution). We call the minimum degree of an optimal community the *optimal min-degree*. Note that, the optimal community is not unique, but the optimal min-degree is unique. It is easy to see that if H is an optimal community to the SCS problem, then the subgraph of G induced by $V(H)$ is also an optimal community. Thus, *we consider only vertex-induced subgraphs in*

the remainder of the chapter, and we denote a subgraph simply by its set of vertices. The notations that are frequently used in this chapter are summarized in Table 3.1.

Table 3.1: Frequently used notations

Notation	Definition
$N_S(u)$	the set of neighbors of u in the subgraph S
$d_S(v)$	the degree of vertex v in the subgraph S
$d_{\max}(S)$	maximum degree of the subgraph S
$d_{\min}(S)$	minimum degree of the subgraph S
$\mathsf{cn}(v)$	core number of vertex v in G
$dist_S(u, v)$	the shortest distance between u and v in the subgraph S
$\mathcal{C}_{C,R}$	the set of all feasible communities X s.t. $C \subseteq X \subseteq C \cup R$
$\tilde{k}$ and $\hat{k}$	lower bound and upper bound of the optimal min-degree
$C^{\leq \tau}$	the set of vertices of degree $\leq \tau$, *i.e.*, $\{u \in C : d_C(u) \leq \tau\}$

Example 1. *Consider the graph in Figure 3.1, and suppose the SCS query is* $q = v_1$ *and* $[\ell, h] = [3, 5]$. *Then,* $H_1 = \{v_0, v_1, v_2\}$ *is a feasible community which has minimum degree* 2, *while* $H_2 = \{v_0, v_1, v_2, v_3, v_4\}$ *is an optimal community which has minimum degree* 3.

We prove in the theorem below that the SCS problem is NP-hard.

Theorem 1. *The SCS problem is NP-hard.* [2]

Proof. We prove that the decision version of the SCS problem is NP-complete, by reducing from the k-clique problem which is NP-complete (Lewis, 1983). The decision SCS problem is as follows: given a graph $G = (V, E)$, a query vertex $q \in V$, a size constraint $[\ell, h]$ and an integer k, determine whether G has a subgraph g such that (1) g is connected and contains q, (2) $\ell \leq |V(g)| \leq h$, and (3) the minimum degree of g is at least k. It is obvious that the decision problem is in NP.

Now, given an instance of the k-clique problem which takes a graph $G = (V, E)$ and an integer k as input and aims to determine whether G has a k-clique (*i.e.*, a complete subgraph with k vertices), we reduce it into a decision SCS problem as follows. We add a dummy vertex v_q, and an edge between v_q and every vertex of V. Denote the resulting graph as G', *i.e.*, $V(G') = V \cup \{v_q\}$ and $E(G') = E \cup \{(v_q, v) \mid v \in V\}$. The query of our decision SCS problem consists of a query vertex v_q, size constraint $[k + 1, k + 1]$, and minimum-degree threshold k. It is easy to verify that $C \subseteq V$ is a k-clique if and only if $C \cup \{v_q\}$ is a solution to the decision SCS problem. Thus, the decision SCS problem is NP-complete, and the (optimization version of) SCS problem is NP-hard. □

3.3 A Baseline Approach

In this section, we present a baseline approach for the SCS problem. It computes an optimal community by *enumerating* all feasible communities and identifying the one with the largest min-degree.

[2] Note that the proof in (Sozio and Gionis, 2010) reduces from the Steiner tree problem, and thus does not work for the special case that there is only one query vertex.

Algorithm 1: SC-Enum$(G, q, [\ell, h])$

```
1  Heuristically compute a feasible community H;   /* e.g., invoke GreedyF */;
2  k̃ ← minimum degree of G[H];   /* Lower bound of optimal min-degree */;
3  Compute core number cn(·) for all vertices in G;
4  k̂ ← min{cn(q), h − 1};        /* Upper bound of optimal min-degree */;
5  if k̃ < k̂ then
6      Remove from G all vertices v satisfying cn(v) ≤ k̃;
7      Enum({v_q}, V(G) \ {v_q});
8  return H;

   Procedure Enum(C, R)
9  if |C| ∈ [ℓ, h] and d_min(C) > k̃ then
10     k̃ ← d_min(C); H ← C;
11 if |C| < h and R ≠ ∅ then
12     v ← a vertex from R;
13     Enum(C ∪ {v}, R \ {v});
14     Enum(C, R \ {v});
```

The pseudocode of the enumeration procedure Enum is shown in Lines 9–14 of Algorithm 1. Given a partial solution C and a candidate set R of vertices such that $C \cap R = \emptyset$, Enum aims to enumerate all feasible communities X such that $C \subseteq X \subseteq C \cup R$; denote the set of all such feasible communities by $\mathcal{C}_{C,R}$. Enum in addition maintains H, the currently found best community (*i.e.*, the one with the largest min-degree among all enumerated feasible communities), and $\tilde{k}$, the minimum degree of $G[H]$, which is a lower bound of the optimal min-degree. If C is a feasible community (*i.e.*, $|C| \in [\ell, h]$) and the minimum degree of $G[C]$ is larger than $\tilde{k}$ (Line 9), then Enum updates H and $\tilde{k}$ (Line 10); note that, C is assumed to always contain q, and for presentation simplicity, we also assume that C is connected. Then, to enumerate all feasible communities $\mathcal{C}_{C,R}$ for the instance (C, R), Enum partitions the enumeration space into $(C \cup \{v\}, R \setminus \{v\})$ and $(C, R \setminus \{v\})$ for an arbitrarily chosen vertex $v \in R$ and conducts a recursion on each of the two enumeration subspaces (Lines 12–14); note that, $\mathcal{C}_{C \cup \{v\}, R \setminus \{v\}} \cap \mathcal{C}_{C, R \setminus \{v\}} = \emptyset$ and $\mathcal{C}_{C \cup \{v\}, R \setminus \{v\}} \cup \mathcal{C}_{C, R \setminus \{v\}} = \mathcal{C}_{C,R}$. Moreover, as the feasible communities have a size upper bound h, we can stop the recursion if $|C| \geq h$ (Line 11). It can be verified by induction that invoking Enum with $C = \{q\}$ and $R = V(G) \setminus \{q\}$ successfully enumerates all feasible communities of G.

Example 2. *For the graph and query in Example 1, Figure 3.2 illustrates a part of the search tree of the enumeration procedure* Enum*, where the root is the query vertex* v_1*. Initially,* $C = \{v_1\}$ *and* $R = V(G) \setminus \{v_1\}$*. In the first level,* v_0 *is selected from* R *and two branches/recursions are generated: the left branch moves* v_0 *from* R *to* C*, and the right branch discards* v_0*. For the first branch, we have* $C = \{v_1, v_0\}$ *and* $R = V(G) \setminus \{v_1, v_0\}$*, and we proceed forward by selecting the next unvisited vertex in* R*,* i.e.*,* v_2*. By moving* v_2 *from* R *to* C*, we get* $C = \{v_1, v_0, v_2\}$*, which is a feasible community with minimum degree* 2*. Thus we update* $\tilde{k}$ *as* 2 *and* H *as* $\{v_1, v_0, v_2\}$*.*

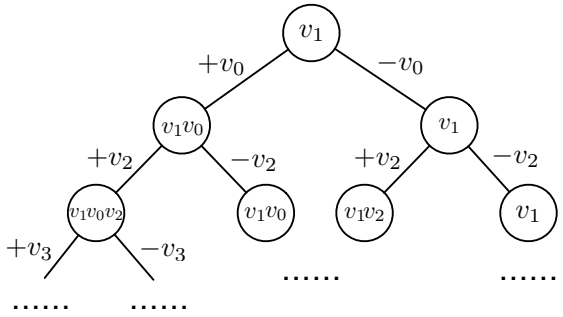


Figure 3.2: An example search tree

The search continues in a similar way until all the feasible communities are explored. Finally, we get the optimal solution $H = \{v_1, v_0, v_2, v_3, v_4\}$ *with minimum degree* 3.

The time complexity of Enum is in the order of n^h, where n is the number of vertices in G and h is the size upper bound. To reduce n, we preprocess the graph G before invoking Enum. The pseudocode is given in Lines 1–8 of Algorithm 1. We first heuristically compute a feasible community H, *e.g.*, by invoking the heuristic algorithm GreedyF (Sozio and Gionis, 2010) (Line 1), and set the lower bound $\tilde{k}$ of the optimal min-degree as the minimum degree of $G[H]$ (Line 2). We also compute the upper bound $\hat{k}$ of the optimal min-degree as $\min\{\mathsf{cn}(q), h-1\}$ (Line 3–4); it is easy to see that the optimal min-degree cannot be larger than $h-1$ and also cannot be larger than $\mathsf{cn}(q)$. If $\tilde{k} = \hat{k}$, then H is already the optimal community. Otherwise, we shrink the graph G by removing all unpromising vertices (*i.e.*, the vertices v with $\mathsf{cn}(v) \leq \tilde{k}$), and then invoke Enum on the reduced graph (Lines 5–6). The overall algorithm is denoted SC-Enum. Note that, to extend SC-Enum to handle multiple query vertices, we would need to (1) initially put all query vertices into the partial solution C at Line 7 and (2) check at Line 9 whether $G[C]$ is connected (if $G[C]$ is not connected, then we do not update $\tilde{k}$ and H at Line 10).

Limitations of the Baseline Approach. Despite the preprocessing of SC-Enum which reduces the input graph size, SC-Enum is still inefficient in processing large graphs. This is mainly due to the following three limitations of Enum.

(1) Large Candidate Set. Enum considers all vertices of R one by one, and in each recursion, it reduces the size of R only by one. Thus, the size of the candidate set R remains large during the recursion. However, given the current lower bound $\tilde{k}$ of the optimal min-degree, many vertices of R will not be part of any feasible community of $\mathcal{C}_{C,R}$ that has min-degree larger than $\tilde{k}$; thus, these vertices can be removed from R to reduce the search space.

(2) No Pruning based on Upper Bounds. For a given instance (C, R), Enum needs to enumerate all feasible communities of $\mathcal{C}_{C,R}$, *i.e.*, all subgraphs X such that $C \subseteq X \subseteq C \cup R$ and $\ell \leq |X| \leq h$. However, if we can compute an upper bound on the largest min-degree among all feasible communities in $\mathcal{C}_{C,R}$, then we can terminate/prune this instance if the upper bound is no larger than $\tilde{k}$. In this way, the search space will be significantly reduced.

(3) Naive Branching Rule. Enum arbitrarily selects one vertex $v \in R$ and then generates two recursions/branches, $(C \cup \{v\}, R \setminus \{v\})$ and $(C, R \setminus \{v\})$. This may

generate a large number of recursions. Also, the ordering of generating the subinstances of Enum (*i.e.*, via selecting the vertex of R to branch) may affect the speed of tightening the lower bound $\tilde{k}$, and thus affect the search tree size.

3.4 A Branch-Reduce-and-Bound Approach

In this section, we propose a branch-reduce-and-bound algorithm SC-BRB for solving the SCS problem. SC-BRB specifically addresses the three limitations of Enum by proposing branching techniques, reducing techniques, and upper bounding techniques. Note that designing branching, reducing, and bounding techniques is the standard approach (and usually the only approach) for exactly solving NP-hard problems (Woeginger, 2003).

In the following, we present reducing techniques in Section 3.4.1, upper bounding techniques in Section 3.4.2, and branching techniques in Section 3.4.3. Finally, Section 3.4.4 presents the overall algorithm SC-BRB.

3.4.1 Reducing Techniques

Given an instance (C, R) of Enum, we propose reduction rules to reduce the size of R by either removing unpromising vertices from R or greedily moving promising vertices from R to C. Recall that, given an instance (C, R), we aim to find the feasible community in $\mathcal{C}_{C,R}$ that has the largest min-degree if this min-degree is larger than $\tilde{k}$. That is, among all feasible communities in $\mathcal{C}_{C,R}$, we are only interested in the ones that have min-degree larger than $\tilde{k}$, as the currently found best community has min-degree $\tilde{k}$.

The first reduction rule prunes a vertex v from R based on its degree in $C \cup R$ or its degree in $C \cup \{v\}$.

Reduction Rule 1 (Degree-based Reduction)**.** Given an instance (C, R) and any vertex $v \in R$, if $\min\{d_{C\cup R}(v), d_{C\cup\{v\}}(v) + h - |C| - 1\} \leq \tilde{k}$, then we can discard v from R, where $d_{C\cup R}(v)$ is the degree of v in the subgraph $G[C \cup R]$.

Proof. As the degree of a vertex is a monotonically increasing property, it holds that $d_X(v) \leq d_{C\cup R}(v)$ for all feasible communities X of $\mathcal{C}_{C,R}$ that contain v, *i.e.*, $C \cup \{v\} \subseteq X \subseteq C \cup R$. Thus, if $d_{C\cup R}(v) \leq \tilde{k}$, then all feasible communities of $\mathcal{C}_{C,R}$ that contain v have min-degree at most $\tilde{k}$, and hence v can be safely discarded.

Similarly, for any feasible community $X \in \mathcal{C}_{C,R}$ that contains v, the degree $d_X(v)$ of v in X is upper bounded by $d_{C\cup\{v\}}(v) + h - |C| - 1$ even if we further add, to $C \cup \{v\}$, $h - |C| - 1$ vertices that are all adjacent to v. Thus if this upper bound is no larger than $\tilde{k}$, then all such feasible communities $X \in \mathcal{C}_{C,R}$ that contain v will have minimum degree at most $\tilde{k}$, and hence v can be discarded. □

The next reduction rule is based on a lower bound, as shown in the lemma below, on the size (*i.e.*, number of vertices) of a graph that has minimum degree k and diameter D. The *diameter* of a graph is the largest value among all pair-wise shortest distances, *i.e.*, $\max\{dist_G(u, v) \mid u, v \in V(G)\}$, where $dist_G(u, v)$ denotes the shortest distance between u and v in G.

Lemma 1. *Any graph of minimum degree $k \geq 1$ and diameter $D \geq 1$ must have at least $n(k, D)$ vertices, where*

$$n(k,D) = \begin{cases} k + D & \text{if } 1 \leq D \leq 2 \text{ or } k = 1 \quad (3.1a) \\ k + D + 1 + \lfloor \frac{D}{3} \rfloor (k-2) & \text{otherwise} \quad (3.1b) \end{cases}$$

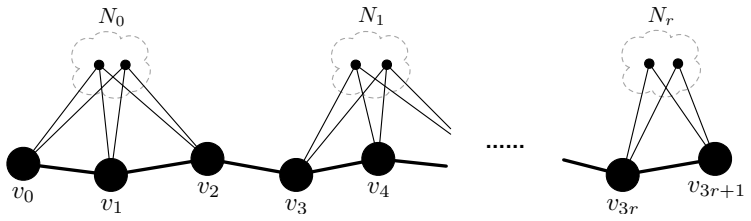
Moreover, this bound is tight; that is, for every $k \geq 1$ and every $D \geq 1$, there exists a graph with $n(k, D)$ vertices that has minimum degree k and diameter D.

Proof. Let g be a graph of minimum degree $k \geq 1$ and diameter $D \geq 1$. Let $P = (v_0, v_1, \ldots, v_D)$ be a path in the graph such that the shortest distance between v_0 and v_D is D. Note that, such a path P must exist as the diameter of the graph is D.

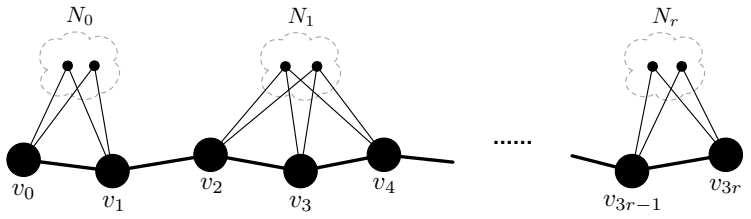
We first consider the case that $D \geq 3$ and $k \geq 2$. Let N_j be the set of v_{3j}'s neighbors that are not in P for $j = 0, \ldots, \lfloor \frac{D}{3} \rfloor - 1$, and $N_{\lfloor \frac{D}{3} \rfloor}$ be the set of v_D's neighbors that are not in P. Then, it holds that $N_j \cap N_{j'} = \emptyset, \forall j \neq j'$; this is because the shortest distance between v_0 and v_D would otherwise be smaller than D. In addition, as the minimum degree of the graph is k, we have the following facts (see Figure 3.3).

- $|N_0| \geq k - 1$.
- $|N_j| \geq k - 2$, for $0 < j < \lfloor \frac{D}{3} \rfloor$.
- $|N_{\lfloor \frac{D}{3} \rfloor}| \geq k - 1$.

As a result, the number of vertices in the graph is at least $D + 1 + \sum_{j=0}^{\lfloor \frac{D}{3} \rfloor} |N_j| \geq k + D + 1 + \lfloor \frac{D}{3} \rfloor (k-2) = n(k, D)$; note that for the case that $D \leq 2$, this is $D + 1 + \sum_{j=0}^{\lfloor \frac{D}{3} \rfloor} |N_j| = D + 1 + |N_0| \geq k + D$.



(a) Case 1 ($D \bmod 3 \neq 0$)



(b) Case 2 ($D \bmod 3 = 0$)

Figure 3.3: Proof of Lemma 1 ($r = \lfloor \frac{D}{3} \rfloor$)

Note that, the bound $n(k, D) = k + D + 1 + \lfloor \frac{D}{3} \rfloor (k-2)$ is also tight; that is, there exists a graph with $n(k, D)$ vertices and of minimum degree k and diameter

D. Specifically, the graph consists of the path P and sets of vertices $\{N_j\}_{j=0}^{\lfloor \frac{D}{3} \rfloor}$ of minimum sizes as defined above, each N_j is a clique, and there are no crossing edges between N_j and $N_{j'}$ for $j \neq j'$. In addition, vertices in N_j are connected to vertices in P as follows.

- If (D mod 3) $\neq 0$, all vertices in N_j are connected to all vertices in $\{v_i \mid 3j \leq i \leq \min(3j+2, D)\}$, for each $0 \leq j \leq \lfloor \frac{D}{3} \rfloor$ (see Figure 3.3(a));
- Otherwise, N_0 is connected to $\{v_0, v_1\}$, $N_{\lfloor \frac{D}{3} \rfloor}$ is connected to $\{v_{D-1}, v_D\}$, and N_j is connected to $\{v_i \mid 3j-1 \leq i \leq 3j+1\}$ for $0 < j < \lfloor \frac{D}{3} \rfloor$ (see Figure 3.3(b)).

It is easy to verify that for each of the above two cases, the constructed graph has exactly $n(k, D)$ vertices, and has minimum degree k and diameter D.

The cases where $D = 1$ or $D = 2$ are special cases of the above, and can be easily verified. For the case $k = 1$, it suffices to verify that the graph consisting of the path P is a tight lower bound. We omit the details. □

Note that, the lower bound of Equation (3.1b) was stated in (Erdős et al., 1989) for the case $D \geq 2$ and $k \geq 2$, but without detailed proof. Our proof shows that the lower bound of Equation (3.1b) only holds for $D \geq 3$ and $k \geq 2$. Specifically, consider the graph that is a $(k+2)$-vertex clique missing one edge which has minimum degree k and diameter $D = 2$; however, Equation (3.1b) would imply a lower bound of $k + 3$ for $D = 2$ which is incorrect. Thus, we provide a correct and tight lower bound of $k + D$ for the special case $1 \leq D \leq 2$ or $k = 1$.

Moreover, instead of using a closed formula to upper bound the diameter given h and k (*i.e.*, $\lfloor \frac{3h}{k+1} \rfloor - 1$ in (Erdős et al., 1989)) which is not tight, we obtain the upper bound numerically. That is, the diameter is upper bounded by the largest D such that $n(k, D) \leq h$, which can be computed by binary search on D. This is because, $n(k, D)$ monotonically increases with both k and D. For example, for $h = 11$ and $k = 5$, the closed formula of (Erdős et al., 1989) gives an upper bound of 4 for D, while our approach computes the upper bound as 2. Also note that, for a given size upper bound h, the maximum possible diameter D of a feasible community monotonically decreases when k increases. Computing the diameter, however, is computationally expensive. We use the following distance-based reduction, as the shortest distance between any pair of vertices is a lower bound of the diameter.

Reduction Rule 2 (Distance-based Reduction)**.** Given an instance (C, R) and any $v \in R$, if $n(\tilde{k}+1, dist_{C \cup R}(u, v)) > h$, then we can discard v from R, where u is a vertex in C and $n(\cdot, \cdot)$ is the function defined in Lemma 1.

Proof. As the shortest distance is a monotonically decreasing property, it holds that $dist_X(u, v) \geq dist_{C \cup R}(u, v)$ for every feasible community $X \in \mathcal{C}_{C,R}$ that contains v. Thus, the diameter of every feasible community in $\mathcal{C}_{C,R}$ that contains v must be at least $dist_{C \cup R}(u, v)$. According to Lemma 1, if $n(\tilde{k}+1, dist_{C \cup R}(u, v)) > h$, then all feasible communities in $\mathcal{C}_{C,R}$ that contain v must have minimum degree at most $\tilde{k}$, and thus v can be discarded. □

Besides discarding unpromising vertices from R, we can also greedily move promising vertices from R to C. The next reduction rule formulates such a condition.

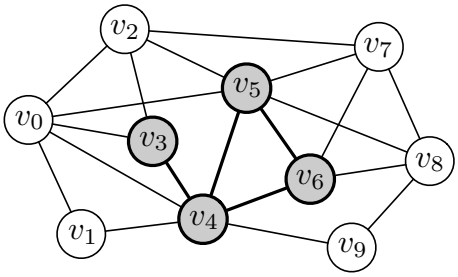


Figure 3.4: A running example for illustrating reduction rules

Reduction Rule 3 (Inclusion-based Reduction)**.** Given an instance (C, R) and any $u \in C$, if $d_{C\cup R}(u) = \tilde{k}+1$, then we can greedily move to C all the vertices in R that are neighbors of u.

Proof. Recall that, for the instance (C, R), our aim is to find a feasible community in $\mathcal{C}_{C,R}$ with minimum degree at least $\tilde{k}+1$. Thus, if $u \in C$ has only $\tilde{k}+1$ neighbors in $C \cup R$, then any feasible community in $\mathcal{C}_{C,R}$ with minimum degree at least $\tilde{k}+1$ must include all of u's neighbors. □

Example 3. *Consider the graph in Figure 3.4, and suppose* $[\ell, h] = [5, 7]$, $C = \{v_3, v_4, v_5, v_6\}$ *(grey-shadowed nodes) and* $R = \{v_0, v_1, v_2, v_7, v_8, v_9\}$ *(white nodes), and* $\tilde{k} = 2$. *Recall that our aim is to find a feasible community in* $\mathcal{C}_{C,R}$ *with minimum degree at least* $\tilde{k}+1 = 3$. *According to Reduction Rule* 1, v_1 *can be discarded because* $d_{C\cup R}(v_1) = 2 \leq \tilde{k}$; *similarly,* v_9 *can be discarded. From Lemma 1, we derive that the diameter of any feasible community with min-degree at least* 3 *is at most* 2; *this is because* $n(3, 3) = 8 > h$. *Thus,* v_8 *can be discarded by Reduction Rule* 2, *since* $dist_G(v_3, v_8) = 3$. *At last, as* $d_{C\cup R}(v_3) = \tilde{k}+1$, *we move all neighbors of* v_3 *in* R *(*i.e., v_0 *and* v_2*) to* C. *Similarly, we move the neighbors of* v_6 *in* R *(*i.e., v_7*) to* C. *As a result, we obtain a feasible community with min-degree* 3 *by the reduction rules,* i.e., $\{v_0, v_2, v_3, v_4, v_5, v_6, v_7\}$.

3.4.2 Upper Bounding Techniques

Let $\mathsf{OptMD}(C, R)$ denote the largest min-degree among all feasible communities of $\mathcal{C}_{C,R}$. In this subsection, we aim to compute upper bounds of $\mathsf{OptMD}(C, R)$ such that the instance (C, R) can be entirely pruned if this upper bound is no larger than $\tilde{k}$. In the following, we first present the general idea of upper bound computation in Section 3.4.2.1, and then present three approaches to computing the upper bound in Sections 3.4.2.2–3.4.2.4. Finally, we put the three upper bounds together in Section 3.4.2.5.

3.4.2.1 General Idea of Upper Bounding

The reduction rules presented in Section 3.4.1 reduce the size of R by either discarding vertices from R or greedily moving vertices from R to C. This is achieved by inspecting individual vertices of C or R, and making local decisions based on properties of the individual vertices. Upper bounding $\mathsf{OptMD}(C, R)$ in contrast aims to

prune the entire instance (C, R), by considering all vertices of C and R altogether. Recall that computing $\mathsf{OptMD}(C, R)$ is to identify the X such that

Condition (1) $C \subseteq X \subseteq C \cup R$,

Condition (2) $\ell \leq |X| \leq h$, and

Condition (3) $\min_{u \in X} d_X(u)$ is maximized.

Let X^* be the best community in $\mathcal{C}_{C,R}$, *i.e.*, satisfying the above three conditions. Note that, computing $\mathsf{OptMD}(C, R)$ is NP-hard, as $\mathsf{OptMD}(\{q\}, V(G) \setminus \{q\})$ is the optimal min-degree of the SCS problem. Thus, we compute an upper bound of $\mathsf{OptMD}(C, R)$ by relaxing condition (2) and/or using upper bounds of $d_{X^*}(u)$. Specifically, we will compute an upper bound of $\min_{u \in C} d_{X^*}(u)$ by *only considering the degrees of vertices of* C, which obviously is an upper bound of $\min_{u \in X^*} d_{X^*}(u)$.

It is worth mentioning that, for our algorithm, it suffices to check whether the minimum degree of X^* is at least $\tilde{k} + 1$. Nevertheless, we present our techniques for computing upper bounds, as computing upper bounds is more general in the sense that it could also be potentially used in other search paradigms (*e.g.*, best-first search). We will show in Section 3.4.2.5 that, with simple modifications, the upper bounding techniques can be used to efficiently check whether the minimum degree of X^* is at least $\tilde{k} + 1$. Thus, we do not utilize $\tilde{k}$ in the following upper bounding techniques.

3.4.2.2 Degree-based Upper Bound

Firstly, we introduce the degree-based upper bound, denoted by U_d, by considering each vertex $u \in C$ individually and by upper bounding $d_{X^*}(u)$.

Lemma 2 (Degree-based Upper Bound)**.**

$$U_d = \min_{u \in C} \min\{d_{C \cup R}(u), d_C(u) + h - |C|\}$$

is an upper bound of $\mathsf{OptMD}(C, R)$.

Proof. It is easy to verify that for any vertex $u \in C$, the degree of u in X^* is at most $d_C(u) + \min\{h - |C|, d_{R \cup \{u\}}(u)\}$, where $d_C(u) + d_{R \cup \{u\}}(u) = d_{C \cup R}(u)$. Specifically, if $h - |C| < d_{R \cup \{u\}}(u)$, then we consider the best hypothetical scenario that all vertices of $X^* \setminus C$, whose quantity is at most $h - |C|$, are adjacent to u; otherwise, we consider the best hypothetical scenario that all of u's neighbors in R are included in $X^* \setminus C$. Thus, $\mathsf{OptMD}(C, R) = \min_{u \in X^*} d_{X^*}(u) \leq \min_{u \in C} d_{X^*}(u) \leq U_d$. □

The degree-based upper bound is in analogous to the degree-based reduction rule presented in Section 3.4.1. But the degree-based upper bound considers degrees of vertices of C, while the degree-based reduction rule considers degrees of vertices of R.

Example 4. *Consider the graph in Figure 3.5, and suppose* $[\ell, h] = [5, 8]$, $C = \{v_2, v_3, v_5, v_6, v_7\}$ *and* $R = \{v_0, v_1, v_4, v_8, v_9, v_{10}\}$. *We use* $\hat{d}(u)$ *to denote* $d_C(u) + \min\{h - |C|, d_{R \cup \{u\}}(u)\}$, *where* $h - |C| = 3$. *The maximum possible degree of* v_2 *in*

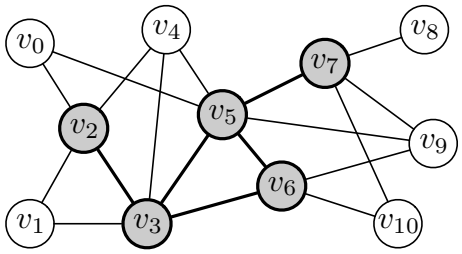


Figure 3.5: A running example for illustrating upper bounds

X^* *is* 4, i.e., $\hat{d}(v_2) = 1 + \min\{3, 3\} = 4$. *For* v_3, *as it has only* 2 *neighbors outside* C, i.e., $d_{R\cup\{v_3\}}(v_3) = 2$, *we have* $\hat{d}(v_3) = 3 + \min\{3, 2\} = 5$. *Similarly, we derive* $\hat{d}(v_5) = 3 + \min\{3, 3\} = 6$, $\hat{d}(v_6) = 2 + \min\{3, 2\} = 4$ *and* $\hat{d}(v_7) = 1 + \min\{3, 3\} = 4$. *Finally, we get* $U_d = 4$.

Time Complexity. In the implementation, we incrementally maintain the degrees $d_{C\cup R}(u)$ and $d_{C\cup\{u\}}(u)$ for each vertex $u \in C$. Thus, the time complexity of computing U_d is $\mathcal{O}(|C|)$.

3.4.2.3 Neighbor Reconstruction-based Upper Bound

The degree-based upper bound U_d ignored the degrees of vertices of R and assumed that the degrees of vertices of $X^* \setminus C \subseteq R$ could be arbitrarily large. Here, we design the neighbor reconstruction-based upper bound, by explicitly considering the degrees of vertices of R. Specifically, we define the *neighbor reconstruction* problem: given a graph g, b vertices $v_1, \ldots, v_b \notin V(g)$, and b non-negative integers $d_1, \ldots, d_b$, how to add edges between $V(g)$ and $\{v_1, \ldots, v_b\}$ without parallel edges, such that degrees of $v_1, \ldots, v_b$ are $d_1, \ldots, d_b \in [0, |V(g)|]$, respectively, and the minimum degree among vertices of g in the resulting graph is maximized. Intuitively, by letting $g = G[C]$, $b = h - |C|$, $\{v_1, \ldots, v_b\}$ be the b vertices of R with the most number of neighbors in C, and d_i be the number of v_i's neighbors in C, then the minimum degree obtained by neighbor reconstruction is an upper bound of $\mathsf{OptMD}(C, R)$.

We propose a greedy approach to solve the neighbor reconstruction problem. As the objective is to maximize the minimum degree among vertices of g in the reconstructed graph, v_i intuitively should be connected to vertices of g that have the smallest degrees. Thus, we process vertices $\{v_1, \ldots, v_b\}$ in an arbitrary order, and when processing v_i, we connect v_i to the d_i vertices in $V(g)$ that currently have the smallest degrees; note that, the degrees of $V(g)$ dynamically increase when new edges are added.

The pseudocode of neighbor reconstruction-based upper bound is shown in Algorithm 2, which is self-explanatory by following the above discussions.

Lemma 3 (Neighbor Reconstruction-based Upper Bound)**.** *Algorithm 2 returns a valid upper bound of* $\mathsf{OptMD}(C, R)$.

Proof. Let $b = h - |C|$, $\{v_1, \ldots, v_b\}$ be the b vertices in R' and $d_i = d_{C\cup\{v_i\}}(v_i)$ for $1 \leq i \leq b$. We first prove that Lines 2–5 of Algorithm 2 solves the neighbor reconstruction problem for C and R'. Let's consider an arbitrary configuration $\mathbb{C}$

Algorithm 2: UB-NeighborReconstruct(C, R)

```
1 R' ← the h − |C| vertices in R that have the largest {d_{C∪{v}}(v) : v ∈ R};
2 for each u ∈ C do d_u ← d_C(u);
3 for each v ∈ R' do
4 |   C' ← the d_{C∪{v}}(v) vertices in C that have the smallest {d_u : u ∈ C};
5 |   for each u ∈ C' do d_u ← d_u + 1;
6 U_nr ← min_{u∈C} d_u;
7 return U_nr;
```

of connecting R' with C that does not follow the strategy in Algorithm 2. Without loss of generality, we assume that the vertices of R' are processed in the order of $v_1, \ldots, v_b$ at Line 3. Let v_i be the first vertex in this order such that $\mathbb{C}$ processes v_i differently than Algorithm 2. Let $\{d_u : u \in C\}$ be the degrees of vertices of C immediately before processing v_i; we simply refer to d_u as the degree of u. Let C' be the d_i vertices in C with the smallest degrees. Then, there must exist a vertex $u' \in C'$ and a vertex $u \in C \setminus C'$ such that $(v_i, u') \notin \mathbb{C}$, $(v_i, u) \in \mathbb{C}$, and $d_{u'} < d_u$. Note that, we do not distinguish vertices of the same degree; that is, if $d_{u'} = d_u$, then we consider $(\mathbb{C} \setminus \{(v_i, u)\}) \cup \{(v_i, u')\}$ to be the same as $\mathbb{C}$. Let $d^*_{u'}$ and d^*_u, respectively, be the final degrees of u' and u obtained by $\mathbb{C}$. We consider two cases.

- If $d^*_{u'} < d^*_u$, then $(\mathbb{C} \setminus \{(v_i, u)\}) \cup \{(v_i, u')\}$ must have a no smaller min-degree than $\mathbb{C}$.
- If $d^*_{u'} \geq d^*_u$, then there must exist another vertex $v_j \in R'$ with $j > i$ such that $(v_j, u') \in \mathbb{C}$ and $(v_j, u) \notin \mathbb{C}$. Thus, $(\mathbb{C} \setminus \{(v_i, u), (v_j, u')\}) \cup \{(v_i, u'), (v_j, u)\}$ must have the same min-degree as $\mathbb{C}$.

In either case, we can resolve all the inconsistencies between $\mathbb{C}$ and the strategy of Algorithm 2 for v_i's edges, while not introducing any inconsistencies for $\{v_j : j < i\}$'s edges. So on so forth, we can resolve all the inconsistencies between $\mathbb{C}$ and the strategy of Algorithm 2 while not decreasing the minimum degree. Thus, Algorithm 2 computes a configuration with the largest min-degree.

Let $R'' = \{v'_1, \ldots, v'_a\}$ be the vertices of $X^* \setminus C$ where $a \leq b$. Obviously, running Lines 3–5 of Algorithm 2 by replacing R' with R'', we obtain a valid upper bound of $\mathsf{OptMD}(C, R)$ based on the above arguments. As the degrees (*i.e.*, $d_{C\cup\{v\}}(v)$) of vertices of R' are no smaller than that of R'', Algorithm 2 returns a valid, albeit maybe larger, upper bound of $\mathsf{OptMD}(C, R)$. □

Example 5. *Consider the same setting as Example 4, Figure 3.6 illustrates the steps of computing U_{nr} by Algorithm 2. The initial degrees d_u of vertices of C are $\{v_2 : 1, v_3 : 3, v_5 : 3, v_6 : 2, v_7 : 1\}$ and the vertices of R with their numbers of neighbors to be reconstructed are $\{v_0 : 2, v_1 : 2, v_4 : 3, v_8 : 1, v_9 : 3, v_{10} : 2\}$. The first step is to select the $h - |C| = 3$ vertices of R that have the largest number of neighbors to be reconstructed, and $R' = \{v_4, v_9, v_0\}$. Then, we process the three vertices of R' sequentially. In the first iteration, we process v_4 by connecting it to the $d_{C\cup\{v_4\}}(v_4) = 3$ vertices of C that currently have the smallest degrees, and*

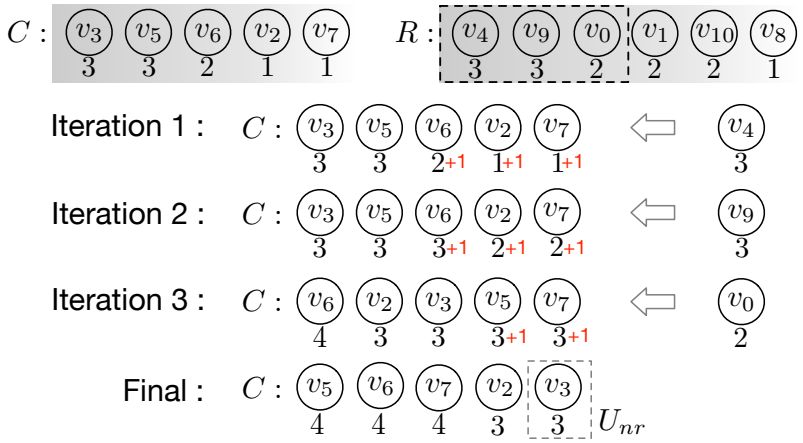


Figure 3.6: Neighbor reconstruction-based upper bound

$C' = \{v_2, v_6, v_7\}$; *after processing* v_4, *the degrees of vertices of* C *become* $\{v_2 : 2, v_3 : 3, v_5 : 3, v_6 : 3, v_7 : 2\}$. *In second iteration, we process* v_9 *similarly and the degrees of* $C' = \{v_2, v_6, v_7\}$ *increase by one each. In the third iteration, we process* v_0 *and the degrees of* $C' = \{v_5, v_7\}$ *increase by one each. Finally, the minimum degree of vertices of* C *is* 3, *which is returned as the upper bound* U_{nr}.

Time Complexity. Firstly, in our implementation of selecting R' at Line 1 of Algorithm 2, instead of considering all vertices of R which can be of large quantity, we only consider the vertices of R that is adjacent to some vertex of C. Thus, the time complexity of selecting R' is $\mathcal{O}(d_{C,R})$ where $d_{C,R} = \sum_{u \in C} d_{R \cup \{u\}}(u)$; note that, selecting top-k numbers from a list of unsorted numbers can be achieved in linear time regardless of k (Cormen et al., 2022). Secondly, each vertex $v \in R'$ can be processed in $\mathcal{O}(|C|)$ time. Thus, the total time complexity of Algorithm 2 is $\mathcal{O}(d_{C,R} + (h - |C|)|C|)$.

3.4.2.4 Degree Classification-based Upper Bound

In the computation of U_{nr}, we utilized the count of all v_i's neighbors in C, $d_{C \cup \{v_i\}}(v_i)$, to decide the number of edges to be reconstructed between v_i and C. Let $C'' = \{u \in C : d_C(u) \geq U_{nr}\}$ be the set of vertices of C whose degrees in C are already no smaller than U_{nr}. Even if we remove all the edges between R and C'' before the neighbor reconstruction (equivalently, decrease the number of edges to be reconstructed between v_i and C), we intuitively will still get a valid (and likely smaller) upper bound of $\mathsf{OptMD}(C, R)$.

Formally, let $d_{\min}(C)$ and $d_{\max}(C)$ be the minimum degree and maximum degree of $G[C]$, respectively, and let $C^{\leq \tau}$ be the set of vertices of C whose degrees in C are at most τ, *i.e.*, $C^{\leq \tau} = \{u \in C : d_C(u) \leq \tau\}$. We prove a more general lemma in below.

Lemma 4. *For any* $\tau \in [d_{\min}(C), d_{\max}(C)]$, *let* U_{nr}^{τ} *be the result of running Algorithm 2 with the following modifications: remove all edges between* R *and* $C \setminus C^{\leq \tau}$, *and replace* C *with* $C^{\leq \tau}$ *at Lines 4 and 6. Then,* U_{nr}^{τ} *is a valid upper bound of* $\mathsf{OptMD}(C, R)$.

Proof. Let $R' = \{v'_1, \ldots, v'_a\}$ be the vertices of $X^* \setminus C$ where $a \leq h - |C|$. Then, by running Lines 2–6 of Algorithm 2 with the modifications in the statement of the lemma, we actually reconstruct the edges between $C^{\leq\tau}$ and $R' = X^* \setminus C$. Following the proof of Lemma 3, it holds that at Line 6, $U^{\tau}_{nr} = \min_{u \in C^{\leq\tau}} d_u \geq \min_{u \in C^{\leq\tau}} d_{X^*}(u)$, and thus $U^{\tau}_{nr} \geq \mathsf{OptMD}(C, R)$. Note that, $C^{\leq\tau}$ does not change during the execution of the algorithm, despite that d_u increases for some vertices. □

Following Lemma 4, we can compute an upper bound U^{τ}_{nr} for each $\tau \in [d_{\min}(C), d_{\max}(C)]$, and then return the minimum one $\min_{\tau \in [d_{\min}(C), d_{\max}(C)]} U^{\tau}_{nr}$ which is also a valid upper bound of $\mathsf{OptMD}(C, R)$. We call this upper bound as the degree classification-based upper bound. Its pseudocode is shown in Algorithm 3, where Lines 4–9 correspond to Lines 1–6 of Algorithm 2 with the modifications as described in Lemma 4; note that Line 11 should be ignored at the moment.

Algorithm 3: UB-DegreeClassification(C, R)

```
1  U_dc ← ∞;
2  for τ ← d_min(C) to d_max(C) do
3      C^{≤τ} ← {u ∈ C : d_C(u) ≤ τ};
4      R' ← the h − |C| vertices in R that have the largest {d_{C^{≤τ} ∪ {v}}(v) : v ∈ R};
5      for each u ∈ C^{≤τ} do d_u ← d_C(u);
6      for each v ∈ R' do
7          C' ← the d_{C^{≤τ} ∪ {v}}(v) vertices in C^{≤τ} that have the smallest
               {d_u : u ∈ C^{≤τ}};
8          for each u ∈ C' do d_u ← d_u + 1;
9      U^τ_nr ← min_{u ∈ C^{≤τ}} d_u;
10     U_dc ← min{U_dc, U^τ_nr};
11     if U_dc ≤ τ + 1 then break;
12 return U_dc;
```

Corollary 2. The degree classification-based upper bound is tighter than the neighbor reconstruction-based upper bound, *i.e.*, $U_{dc} \leq U_{nr}$ for any instance (C, R).

This directly follows from Lemma 4 and Algorithm 3, as $C^{\leq\tau} = C$ when $\tau = d_{\max}(C)$.

Example 6. *Let's reconsider Example 5. We have* $d_{\min}(C) = 1$, $d_{\max}(C) = 3$, $C^{\leq 1} = \{v_2, v_7\}$, $C^{\leq 2} = \{v_6, v_2, v_7\}$ *and* $C^{\leq 3} = \{v_3, v_5, v_6, v_2, v_7\}$, *as shown in Figure 3.7. For each* $\tau \in [1, 3]$, *the numbers of edges to be reconstructed for vertices of* R (i.e., *the number of neighbors in* $C^{\leq\tau}$) *are shown below the vertices in Figure 3.7. For* $\tau = 1$, *the top-3 vertices of* R *are* $R' = \{v_0, v_1, v_4\}$. *After the neighbor reconstruction operations as done by Lines 5–8 of Algorithm 3,* d_{v_7} *increases to* 3 *and* d_{v_2} *increases to* 2, *and thus* $U^1_{nr} = 2$. *Similarly, we will obtain* $U^2_{nr} = 3$ *and* $U^3_{nr} = 3$; *note that, the computation of* U^3_{nr} *is exactly the same as in Example 5, as* $C^{\leq 3} = C$. *Finally, we get* $U_{dc} = \min\{2, 3, 3\} = 2$.

Optimization. To compute U_{dc}, we may need to run neighbor reconstruction (*i.e.*, Lines 3–10 of Algorithm 3) for $d_{\max}(C)$ times, which in the worst case is $|C| - 1$.

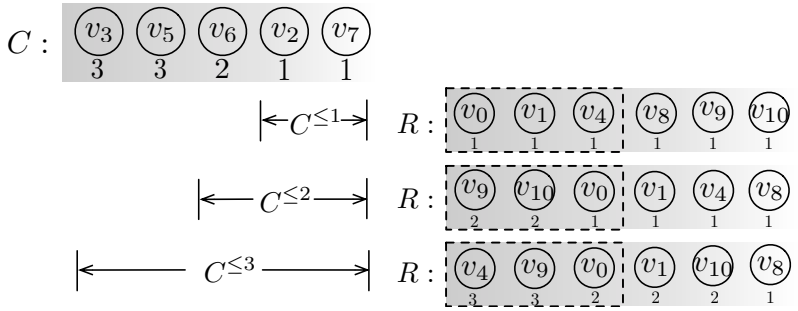


Figure 3.7: Degree classification-based upper bound

Thus, the computation of U_{dc} is costly. In the lemma below, we propose an early stop condition for computing U_{dc}, which corresponds to Line 11 of Algorithm 3.

Lemma 5. *For any* $\tau \in [d_{\min}(C), d_{\max}(C)]$, *if* $\min_{i=d_{\min}(C)}^{\tau} U_{nr}^{i} \leq \tau + 1$, *then* $\min_{i=d_{\min}(C)}^{d_{\max}(C)} U_{nr}^{i} = \min_{i=d_{\min}(C)}^{\tau} U_{nr}^{i}$.

Proof. We prove that if $\min_{i=d_{\min}(C)}^{\tau} U_{nr}^{i} \leq \tau + 1$, then $U_{nr}^{j} \geq \min_{i=d_{\min}(C)}^{\tau} U_{nr}^{i}, \forall \tau < j \leq d_{\max}(C)$, which thus implies the lemma. There are two cases depending on whether $U_{nr}^{\tau} \leq \tau + 1$. Firstly, let's consider the case that $U_{nr}^{\tau} \leq \tau + 1$. Let R^{τ} be the $h - |C|$ vertices in R that are selected for computing U_{nr}^{τ}, and d_u^{τ} be the final degree of vertex $u \in C^{\leq \tau}$ after the neighbor reconstruction for $C^{\leq \tau}$ and R^{τ} by Algorithm 3. Then, $\min_{u \in C^{\leq \tau}} d_u^{\tau} = U_{nr}^{\tau} \leq \tau + 1$. Now, if we further reconstruct $d_{C^{\leq j} \cup \{v\}}(v) - d_{C^{\leq \tau} \cup \{v\}}(v)$ edges between v and $C^{\leq j}$ while avoiding parallel edges, for each $v \in R^{\tau}$, and let d'_u be the final degree of vertex u, then $\min_{u \in C^{\leq j}} d'_u \geq \min\{\tau + 1, \min_{u \in C^{\leq \tau}} d'_u\} \geq \min_{u \in C^{\leq \tau}} d_u^{\tau} = U_{nr}^{\tau}$. Note that, up to now, this actually corresponds to a two-stage neighbor reconstruction between $C^{\leq j}$ and R^{τ} where each $v \in R^{\tau}$ has $d_{C^{\leq j} \cup \{v\}}(v)$ edges to be reconstructed. Now, let's consider a direct (one-stage) approach, and let d''_u be the final degree of vertex u after reconstruction, then $\min_{u \in C^{\leq j}} d''_u \geq \min_{u \in C^{\leq j}} d'_u \geq U_{nr}^{\tau}$ due to the optimality of reconstruction by Algorithm 3 for each τ (or j). Moreover, let R^{j} be the $h - |C|$ vertices in R that are selected for computing U_{nr}^{j}, then the degrees $\{d_{C^{\leq j} \cup \{v\}}(v) : v \in R^{j}\}$ are no smaller than $\{d_{C^{\leq j} \cup \{v\}}(v) : v \in R^{\tau}\}$, and thus $U_{nr}^{j} \geq \min_{u \in C^{\leq j}} d''_u \geq U_{nr}^{\tau}$.

Secondly, let's consider the case that $U_{nr}^{\tau} > \tau + 1$. By following the same argument as above, we can prove that $U_{nr}^{j} \geq \tau + 1$ which implies $U_{nr}^{j} \geq \min_{i=d_{\min}(C)}^{\tau} U_{nr}^{i}$ for all $j \in [\tau + 1, d_{\max}(C)]$. Thus, the lemma holds. □

Example 7. *Let's continue Example 6. After computing* $U_{nr}^{1} = 2$, *it satisfies that* $U_{nr}^{1} \leq 2 = \tau + 1$. *Thus, we can terminate the computation and return* $U_{dc} = 2$ *directly, without computing* U_{nr}^{2} *nor* U_{nr}^{3}.

Time Complexity. As Algorithm 3 in the worst-case runs a modification of Algorithm 2 for $|C| - 1$ times, it is easy to see that the time complexity of Algorithm 3 is $\mathcal{O}\left(|C|\left(d_{C,R} + (h - |C|)|C|\right)\right)$. Note that, by the optimization at Line 11, the number of iterations could be much smaller than $|C| - 1$ in practice.

3.4.2.5 Putting the Upper Bounds Together

To maximally utilize the pruning power, we use all the three upper bounding techniques together and select the minimum one as UB, *i.e.*, $UB = \min\{U_d, U_{nr}, U_{dc}\}$. In the implementation, we first compute U_d, then U_{nr}, and finally U_{dc} in increasing order of their time complexities. Once a computed upper bound is enough to prune the instance (C, R), we terminate the upper bound computation immediately.

3.4.3 Branching Techniques

In this subsection, we propose two branching techniques, one for selecting which vertex to branch on, and another for how to generate the different branches.

Branching Vertex Selection. The lower bound $\tilde{k}$ of the optimal min-degree is critical to the search space size. Thus, we aim to identify feasible communities with large min-degree as early as possible. With this goal in mind, the vertex on which to branch intuitively should satisfy the following two conditions: (1) it should be connected to C; and (2) it should be adjacent to many low-degree vertices in C. To quantify this, we define the *connection score* for vertices of R.

Definition 1 (Connection Score)**.** Given an instance (C, R), the connection score of a vertex $v \in R$ is defined as

$$\delta(v) = \sum_{u \in N_{C \cup \{v\}}(v)} \frac{1}{d_C(u)}$$

Note that, if there is no edge between v and C, then $\delta(v) = 0$.

We choose the vertex of R that has the highest connection score, denoted v^*, to generate branches. Note that this naturally guarantees that C is always a connected subgraph during the recursions.

Domination-based Branching. After choosing v^* as described above, instead of generating two branches $(C \cup \{v^*\}, R \setminus \{v^*\})$ and $(C, R \setminus \{v^*\})$, we propose a domination-based strategy to reduce the number of generated branches.

Definition 2 (Vertex Domination)**.** Given an instance (C, R), vertex $v \in R$ *dominates* $v' \in R$, denoted $v \succeq v'$, if every neighbor of v' (in $C \cup R$) is either a neighbor of v or is v itself.

For instance, for the graph in Figure 3.5, v_4 dominates v_0 and v_1, and v_9 dominates v_8 and v_{10}.

Lemma 6. *Given an instance (C, R) and two vertices $v, v' \in R$, if v dominates v' (*i.e.*, $v \succeq v'$), then there is either a best community in $\mathcal{C}_{C,R}$ (*i.e.*, with minimum degree $\mathsf{OptMD}(C, R)$) that contains both v and v', or a best community in $\mathcal{C}_{C,R}$ that does not contain v'.*

Proof. If there is a best community $H \in \mathcal{C}_{C,R}$ that contains v' but not v, then $H' = (H \cup \{v\}) \setminus \{v'\}$ is also a feasible community in $\mathcal{C}_{C,R}$. Moreover, the minimum degree of H' is no smaller than that of H, as every neighbor of v' in H is also a neighbor of v. Thus, H' is a best community that does not contain v', and the lemma holds. □

Following Lemma 6, after choosing v^* that has the highest connection score, we generate branches as follows. If there is a vertex v' that is dominated by v^*, then we generate three branches $(C \cup \{v^*, v'\}, R \setminus \{v^*, v'\})$, $(C \cup \{v^*\}, R \setminus \{v^*, v'\})$ and $(C, R \setminus \{v^*, v'\})$, which reduces the number of branches from 4, as generated by the naive approach in Algorithm 1, to 3. Moreover, we generalize this idea to the case that v^* dominates multiple vertices. Let $\Phi = \{v_1, \ldots, v_l\}$ be the set of vertices of R that are dominated by v^*. Then, we generate the following branches:

- $(C \cup \{v^*, v_i\}, R \setminus \{v^*, v_1, \ldots, v_i\})$ for $1 \leq i \leq l$.
- $(C \cup \{v^*\}, R \setminus \{v^*, v_1, \ldots, v_l\})$.
- $(C, R \setminus \{v^*, v_1, \ldots, v_l\})$.

3.4.4 The SC-BRB Algorithm

Based on the techniques developed in the previous subsections, we propose the BRB algorithm for solving an instance (C, R). The pseudocode is shown in Algorithm 4, which is similar to Enum in Algorithm 1, but incorporates our newly proposed techniques. We first apply our reduction rules to reduce the size (C, R) (Line 1). If C is a feasible community and its min-degree is larger than $\tilde{k}$, then we update $\tilde{k}$ and H by C (Lines 2–3). Next, if the instance is not pruned by our upper bounding technique (Line 4), then we generate branches as follows. We select the vertex v^* from R that has the highest connection score (Line 5), identify the set $\Phi \subseteq R$ of vertices that are dominated by v^* (Line 6), and order the vertices in Φ in decreasing order based on their connections cores (Line 7). Finally, we generate $|\Phi| + 2$ branches/recursions.

Algorithm 4: BRB(C, R)

```
1  Reduce (C, R) by our reduction rules;
2  if ℓ ≤ |C| ≤ h and d_min(C) > k̃ then
3      k̃ ← d_min(C); H ← C;
4  if |C| < h and R ≠ ∅ and UB(C, R) > k̃ then
5      v* ← the vertex in R with the highest connection score;
6      Φ ← the vertices of R that are dominated by v*;
7      Order the vertices of Φ based on their connection scores, and let the ordered
         vertices be v_1, ..., v_l;
8      for i ← 1, ..., l do
9          BRB(C ∪ {v*, v_i}, R \ {v*, v_1, ..., v_i});
10     BRB(C ∪ {v*}, R \ {v*, v_1, ..., v_l});
11     BRB(C, R \ {v*, v_1, ..., v_l});
```

By replacing the invocation of Enum at Line 7 of Algorithm 1 with BRB, we have our branch-reduce-and-bound algorithm SC-BRB for the SCS problem.

3.5 A Heuristic Approach

In this section, we propose a heuristic algorithm SC-Heu for the SCS problem, which enforces that the size of the returned community is between ℓ and h. We replace GreedyF with SC-Heu in Algorithm 1 for efficiently computing an initial feasible community.

Algorithm 5: SC-Heu(G, q, ℓ, h)

```
1  H ← ∅; k̃ ← 0;
2  if d_G(q) ≥ h − 1 then
3      S ← the ego-network of q;
4      while |S| ≥ ℓ do
5          if |S| ≤ h and d_min(S) > k̃ then
6              k̃ ← d_min(S); H ← S;
7          Delete from S the vertex with the minimum degree;
8  else
9      S ← {q};
10     while |S| < h do
11         v* ← the vertex with the highest connection score to S;
12         S ← S ∪ {v*};
13         if |S| ≥ ℓ and d_min(S) > k̃ then
14             k̃ ← d_min(S); H ← S;
15 return H;
```

The pseudocode of SC-Heu is shown in Algorithm 5. We consider two cases. If the degree of q in G is no smaller than $h-1$ (Line 2), then we adopt the shrinking approach. It starts with the ego-network of q (Line 3) and then iteratively shrinks the graph by removing the vertex with the minimum degree (Line 7); the one with the largest min-degree among all generated feasible communities is returned as the result (Lines 5–6). Here, the ego-network of q in G is the subgraph of G induced by $\{q\} \cup N(q)$. If the degree of q in G is smaller than $h-1$ (Line 8), then we use the expanding approach. It starts with the subgraph $\{q\}$ (Line 9) and then iteratively expands the subgraph by including the vertex with the highest connection score into the subgraph (Line 11–12); the one with the largest min-degree among all generated feasible communities is returned as the result (Lines 13–14).

The time complexity of Algorithm 5 is $\mathcal{O}(m+n\log n)$. Lines 2–7 run in $\mathcal{O}(m)$ time similar to the peeling-based core decomposition algorithm (Batagelj and Zaversnik, 2003). Lines 9–14 run in $\mathcal{O}(m + n\log n)$ time similar to Dijkstra's algorithm for computing single-source shortest paths (Cormen et al., 2022).

3.6 Enumerating All Size-Bounded Communities

In this section, we extend our techniques to the size-bounded community enumeration problem, i.e., enumerating all subgraphs with the largest min-degree that

contain the query vertex q and have at least ℓ and at most h vertices.

Motivation. In the previous sections, we study the problem of size-bounded community search, which only returns one community. However, given a query instance, the size-bounded community may be not unique. Users may have different preferences to different size-bounded communities. For example, when organizing a hiking trip, the owner may hope the attendees to be as more as possible to make more new friends (i.e., the community size should be close to h). When assembling a team for a project, the boss may hope the number of workers to be as less as possible to reduce the cost (i.e., the community size should be close to ℓ). Thus, identifying all size-bounded communities can provide users with more choices.

The pseudo code of our size-bounded community enumeration algorithm is shown in Algorithm 6. Firstly, we compute the optimal min-degree $\tilde{k}$ by invoking SC-BRB (Line 1). Then, we reduce the graph to its $\tilde{k}$-core (Line 2), since each community must have the minimum degree of $\tilde{k}$. Next, we enumerate all communities by invoking the procedure BRB. In BRB, we first apply our reduction rules to reduce the size (C, R) (Line 6). If C is a feasible community and its min-degree equals to $\tilde{k}$, we add it into $\mathbb{C}$. Next, if the instance is not pruned by our upper bounding technique (Line 9), we select the vertex v^* from R that has the highest connection score (Line 10). Then, BRB partitions the enumeration space into $(C \cup \{v^*\}, R \setminus \{v^*\})$ and $(C, R \setminus \{v^*\})$ for the vertex v^* and conducts a recursion on each of the two enumeration subspaces (Lines 11-12). Note that the domination-based branching technique used in SC-BRB cannot be applied in SCE-BRB since Lemma 6 does not hold for the community enumeration problem. Besides, our reduction rules proposed in Section 3.4.1 need minor modifications. For the **Reduction Rule 1**, "$\leq$" should be changed to "$<$". For the **Reduction Rule 2** and the **Reduction Rule 3**, "$\tilde{k}+1$" should be changed to "$\tilde{k}$".

Algorithm 6: SCE-BRB (G, q, ℓ, h)

```
1  k̃ ← the optimal min-degree by invoking SC-BRB;
2  Reduce G to its k̃-core;
3  ℂ ← ∅;
4  Enum({v_q}, V(G) \ {v_q});
5  return ℂ;

   Procedure BRB (C, R)
6  Reduce (C, R) by our reduction rules;
7  if ℓ ≤ |C| ≤ h and d_min(C) = k̃ then
8      ℂ ← ℂ ∪ {C};
9  if |C| < h and R ≠ ∅ and UB(C, R) ≥ k̃ then
10     v* ← the vertex in R with the highest connection score;
11     Enum(C ∪ {v*}, R \ {v*});
12     Enum(C, R \ {v*});
```

3.7 Experiments

In this section, we evaluate the effectiveness and efficiency of the proposed algorithms.

Algorithms. For the size-bounded community search problem, we compare the following algorithms.

- GreedyF and GreedyD: the two heuristic algorithms proposed in (Sozio and Gionis, 2010). Note that, it is straightforward to extend these two algorithms to consider the size lower bound ℓ.
- BS: the algorithm proposed in (Ma et al., 2019) for computing the size-constrained k-core over edge-weighted graphs. We adapt BS to our SCS problem by enumerating k and h.
- PSA: the progressive algorithm proposed in (Li et al., 2019) for (approximately) computing a minimum k-core. We will describe how to adapt PSA in Section 3.7.3.
- SC-Enum: our baseline algorithm presented in Algorithm 1.
- SC-BRB: our algorithm proposed in Section 3.4.4.

For the size-bounded community enumeration problem, we compare the following algorithms.

- SCE-Enum: the baseline algorithm of enumerating all size-bounded communities without applying reduction rules and upper bounding techniques.
- SCE-BRB: our algorithm proposed in Section 3.6.

All algorithms are implemented in C++ and run in main memory.

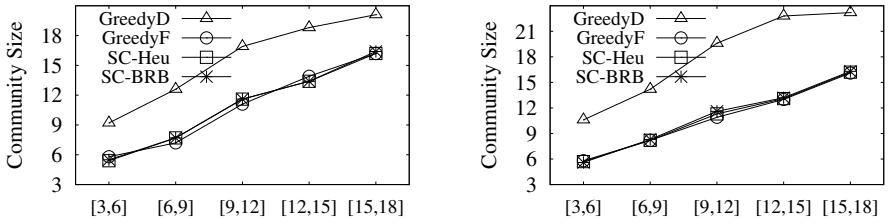
Table 3.2: Statistics of real datasets

Dataset	n	m	d_{avg}	$d_{\max}$	$k_{\max}$
Email	36,692	183,831	10.02	1,383	43
HepPh	34,546	420,877	24.36	846	30
DBLP	317,080	1,049,866	6.62	343	113
YouTube	1,134,890	2,987,624	5.26	28,754	51
Google	875,713	4,322,051	9.87	6,332	44
BerkStan	685,230	6,649,470	19.40	84,230	201
Gplus	107,614	12,238,285	227.44	20,127	752
Flickr	1,715,254	15,551,249	18.13	27,224	568
UK2002	18,459,128	261,556,721	28.33	194,955	943
Webbase	118,142,155	1,019,903,190	17.26	816,127	1,506

Datasets. We evaluate the algorithms on ten real graphs. UK2002 and Webbase are downloaded from WebGraph (Boldi and Vigna, 2004), while all the other graphs are downloaded from SNAP (Leskovec and Krevl, 2014). For each graph, we removed

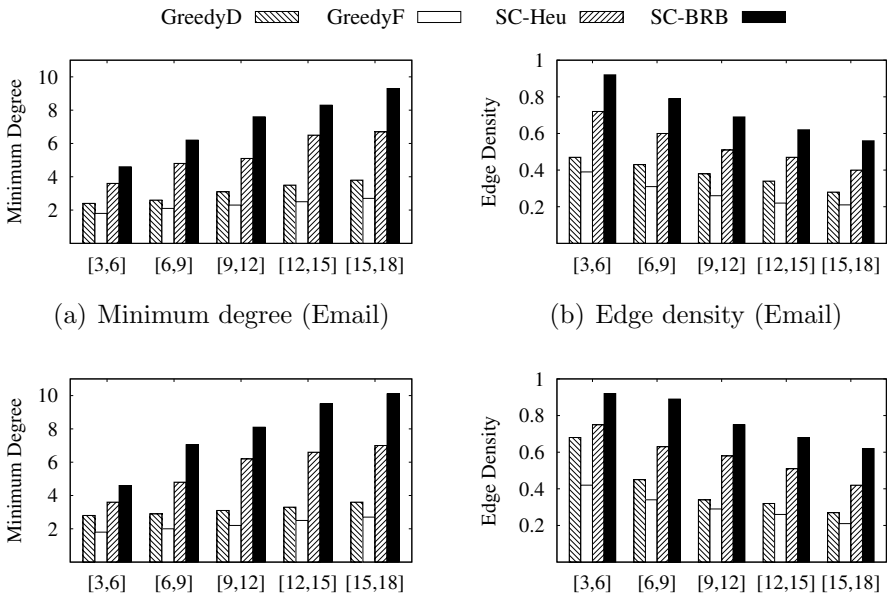
self-loops, parallel edges, as well as the direction of edges. Statistics of the graphs are shown in Table 3.2, where the graphs are listed in increasing order regarding their numbers of edges; $k_{\max}$ is the maximum k such that the graph contains a non-empty k-core.

Besides real graphs, we also generate synthetic graphs to evaluate the efficiency of the algorithms. Specifically, we generated four power-law graphs PL1, PL2, PL3, PL4 by GTgraph (Bader and Madduri, 2006), and four graphs SBM1, SBM2, SBM3, SBM4 by the stochastic block model (Decelle et al., 2011). All the synthetic graphs have 10^6 vertices, and we vary the number of edges from 10^6 (*i.e.*, PL1 and SBM1) to 10^9 (*i.e.*, PL4 and SBM4) with an increasing factor of 10. For the stochastic block model graphs, the number of communities is set as 10^4, and thus each community contains 100 vertices.



(a) Email (b) DBLP

Figure 3.8: Result size

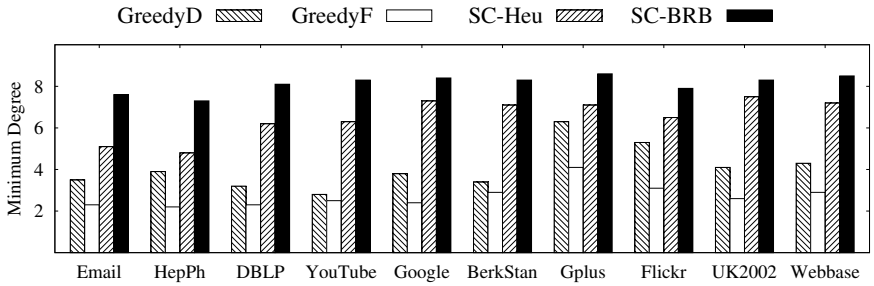


(a) Minimum degree (Email) (b) Edge density (Email)

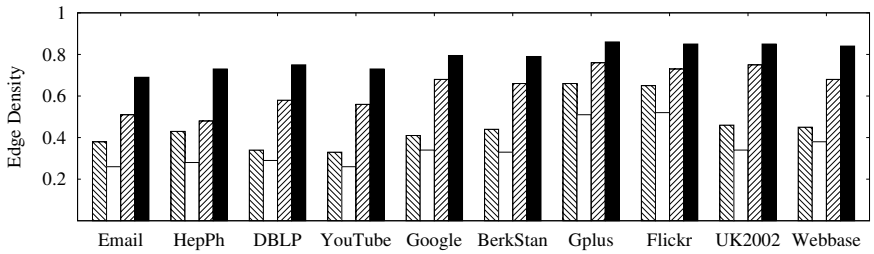
(c) Minimum degree (DBLP) (d) Edge density (DBLP)

Figure 3.9: Result quality by varying $[\ell, h]$

Parameters and Query Generation. We vary the size upper bound h from 6 to 18 with increment 3. For each h, we set the size lower bound ℓ to be $h - 3$. That



(a) Minimum degree



(b) Edge density

Figure 3.10: Result quality on all graphs ($[\ell, h] = [9, 12]$)

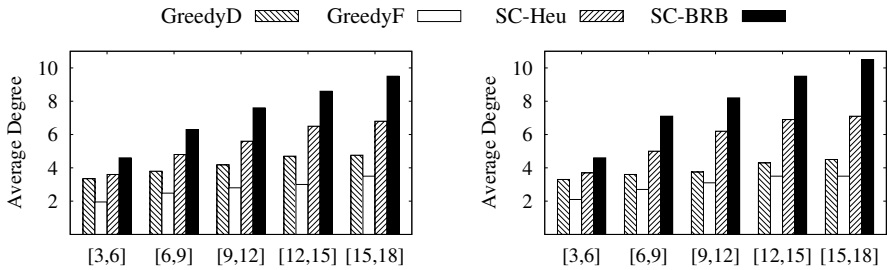
is, the size constraint $[\ell, h]$ is selected from $\{[3, 6], [6, 9], [9, 12], [12, 15], [15, 18]\}$. For each size constraint $[\ell, h]$, we randomly generate 100 queries, where the query vertex of each query is randomly selected from the set of vertices with core number larger than 5; this is similar to previous studies (Fang et al., 2020b) and is to ensure that there is a meaningful community containing the query vertex. Besides these queries, we also fix one of ℓ and h and vary the other (see Sections 3.7.2 and 3.7.3 for details), and we randomly select query vertices from different parts of the graph (i.e., Dense and Sparse queries in Section 3.7.2).

For each $[\ell, h]$, the average result quality and processing time of the 100 queries are reported. For each testing, we set a time limit of two hours, and if an algorithm does not finish within the time limit, we record its running time as inf. All experiments are conducted on a machine with an Intel Core-i7 3.20GHz CPU and Ubuntu system.

3.7.1 Effectiveness of Our Algorithms

In this subsection, we evaluate the effectiveness of our exact algorithm SC-BRB against the existing heuristic algorithms GreedyF and GreedyD. We exclude BS, as it is an exact algorithm and thus the minimum degrees of its reported communities are the same as that of SC-BRB. We also compared the minimum degree, edge density, and overlap ratio between the ground truth communities and that returned by our algorithm SC-BRB, despite that the goal of SC-BRB is not to recover the ground truth communities.

Result Size. Figure 3.8 reports the sizes of the communities returned by GreedyF, GreedyD, SC-Heu and SC-BRB, where the size constraint $[\ell, h]$ varies from $[3, 6]$ to $[15, 18]$. We can see that the result sizes of GreedyF and both of our algorithms fall



(a) Average degree (Email) (b) Average degree (DBLP)

Figure 3.11: Average degree by varying $[\ell, h]$

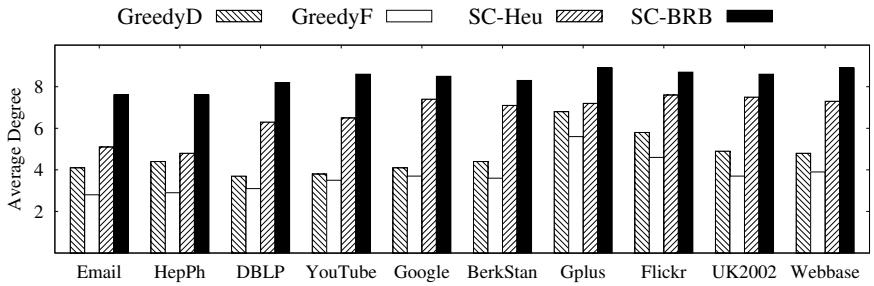


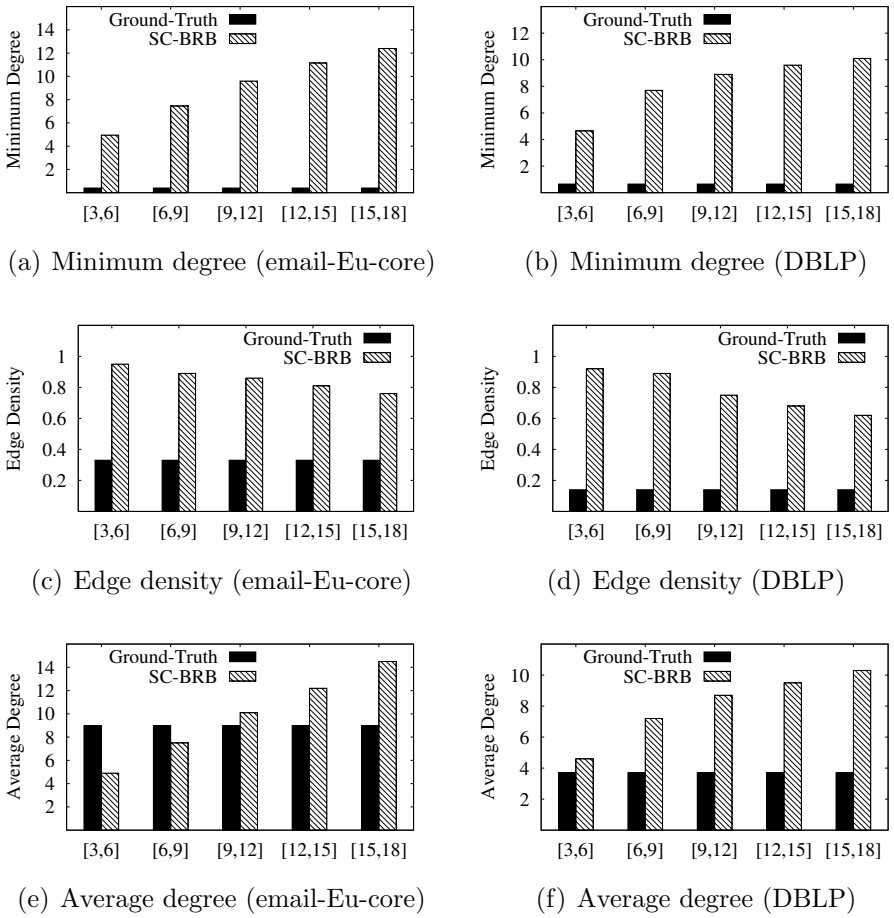
Figure 3.12: Average degree on all graphs ($[\ell, h] = [9, 12]$)

within the range $[\ell, h]$, but GreedyD reports communities with more than h vertices. This is because GreedyF and our algorithms deliberately enforce the size constraint, while GreedyD does not enforce that.

Result Quality. To evaluate the result quality, we report the minimum degree and edge density (*i.e.*, $\frac{2m}{n(n-1)}$) of the extracted communities. For both metrics, the larger the value, the better the quality. The results on Email and DBLP by varying $[\ell, h]$ are shown in Figure 3.9. We can see that GreedyD has a slightly higher result quality than GreedyF, but at the cost of violating the size constraint. Nevertheless, both of our algorithms significantly improve the result quality compared with GreedyD, and our exact algorithm SC-BRB has the highest result quality. The results on all the graphs by fixing $[\ell, h] = [9, 12]$ are shown in Figure 3.10, which have similar trends as in Figure 3.9. Compared with GreedyF, SC-BRB on average increases the minimum degree by a factor of 2.41 (by absolute value 5.05), and improves the edge density by a factor of 2.2.

Evaluate Average Degree for SC-BRB. The average degrees of the communities obtained by the algorithms GreedyD, GreedyF, SC-Heu and SC-BRB are shown in Figure 3.11 and Figure 3.12. We can see that both of our algorithms obtain a much higher average degree than GreedyD and GreedyF, and the results are similar to that for minimum degree and edge density.

Compare with Ground-Truth Communities. Despite that the goal of SCS is not to recover the ground-truth communities, we in this testing compare the communities obtained by SC-BRB with the ground-truth communities that the query vertices belong to. To do that, we use the two graphs email-Eu-core and DBLP that have ground-truth communities. email-Eu-core is a new dataset that is downloaded



(a) Minimum degree (email-Eu-core)

(b) Minimum degree (DBLP)

(c) Edge density (email-Eu-core)

(d) Edge density (DBLP)

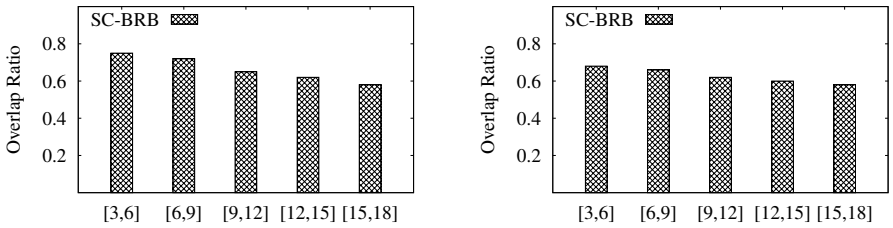
(e) Average degree (email-Eu-core)

(f) Average degree (DBLP)

Figure 3.13: Result quality compared with ground-truth communities

from SNAP, while DBLP has already been tested in our other experiments. email-Eu-core has $1,005$ users (i.e., vertices) in a large European institution, and contains 42 ground-truth communities corresponding to the 42 departments at the research institute. Note that each individual belongs to exactly one department. DBLP is a co-authorship network of researchers in computer science, which contains $317,080$ researchers and $13,477$ communities. Each publication venue defines a ground-truth community, and the authors who published papers in the same journal or conference form a community.

The minimum degree, edge density, and average degree of the communities identified by SC-BRB and that by the ground-truth communities are shown in Figure 3.13. Note that each reported value is the average of 100 random query vertices. We can see that no matter minimum degree, or edge density, or average degree, the quality of the communities identified by our algorithm SC-BRB is much higher than the ground-truth communities. One possible reason is that the ground-truth communities are actually defined in an ad-hoc way as described above, and thus may not correspond to true communities. One exception is the average degree on email-Eu-core for $h \leq 9$. This is because the ground-truth communities are much larger than 9 and thus have an average degree ≥ 9, while the average degree of the communities



(a) Overlap ratio (email-Eu-core) (b) Overlap ratio (DBLP)

Figure 3.14: Overlap ratio with ground-truth communities

identified by SC-BRB cannot be larger than 8 due to the size constraint of $h \leq 9$.

In addition, we also evaluated the overlap ratio (specifically, the percentage of vertices that are in the ground-truth community) between the communities identified by SC-BRB with the ground-truth communities. The results are shown in Figure 3.14. We can see that on average, at least 60% of the vertices in the communities identified by SC-BRB are in the ground-truth communities.

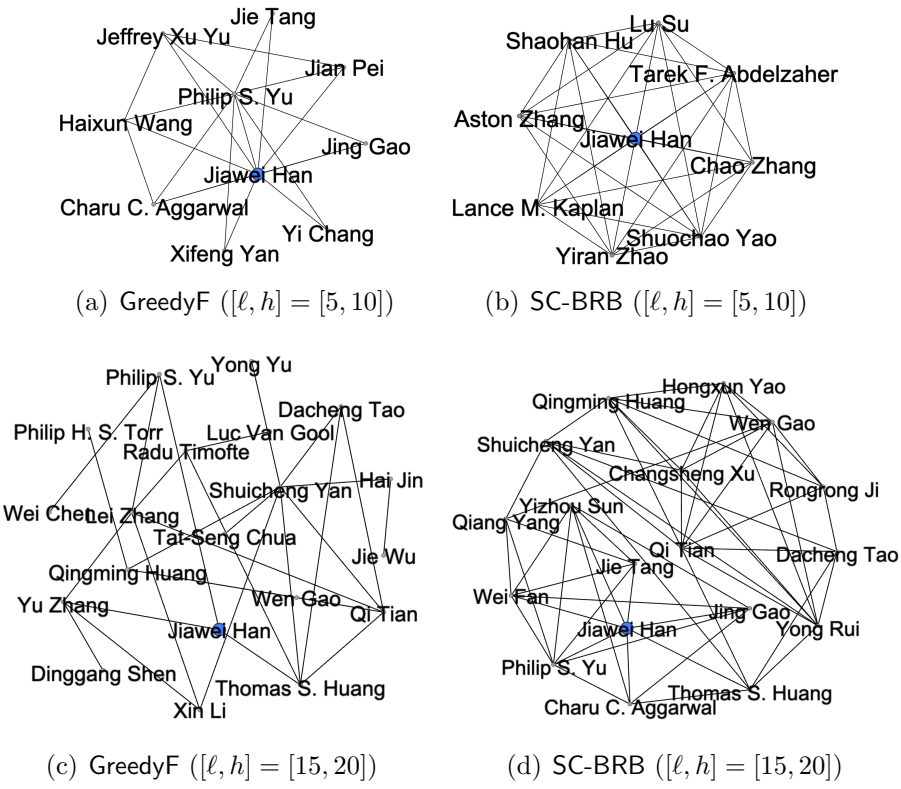
Case Study. We construct another coauthor graph, denoted $\mathsf{DBLP}_{\mathsf{cs}}$, from the raw DBLP dataset [3] for case study. Each vertex corresponds to an author, and two vertices are connected by an edge if the two authors have coauthored at least five papers. The $\mathsf{DBLP}_{\mathsf{cs}}$ graph contains $424,784$ vertices and $888,392$ edges. In the case study, we search for the size-bounded community for "Jiawei Han", a renowned researcher in Data Mining. Firstly, the results returned by GreedyF and SC-BRB for size constraint $[5, 10]$ are shown in Figure 3.15(a) and 3.15(b). We can see that the result by GreedyF is sparse and has minimum degree 2 and edge density 0.48, while the result by our algorithm SC-BRB has minimum degree 7 and edge density 0.91. Figure 3.15(c) and 3.15(d) illustrate the results for size constraint $[15, 20]$. The result by GreedyF has minimum degree 1 and edge density 0.15, while the result by SC-BRB has minimum degree 5 and edge density 0.37.

Suppose Jiawei Han would like to assemble a team to identify and tackle a grand problem in data mining. Then he can issue an SCS query with ℓ and h being specified based on the team size. If the ideal team size is between 5 and 10, then the best team could be the one in Figure 3.15(b) as each member has collaborated with at least 7 other members in the team. If the ideal team size is between 15 and 20, then the best team could be the one in Figure 3.15(d); note that, with the size constraint $[15, 20]$, there is no team such that every member has collaborated with ≥ 6 or ≥ 7 other members.

3.7.2 Efficiency Testings

Against Baseline Algorithms. We first evaluate SC-BRB against baseline algorithms SC-Enum and BS. We adapted BS proposed in (Ma et al., 2019) to solve the SCS problem by enumerating k and $x \in [\ell, h]$ to find the largest k such that there is a k-core of size x. The running time of these algorithms on DBLP and Google by

[3] https://dblp.uni-trier.de/xml/



(a) GreedyF ($[\ell, h] = [5, 10]$) (b) SC-BRB ($[\ell, h] = [5, 10]$)

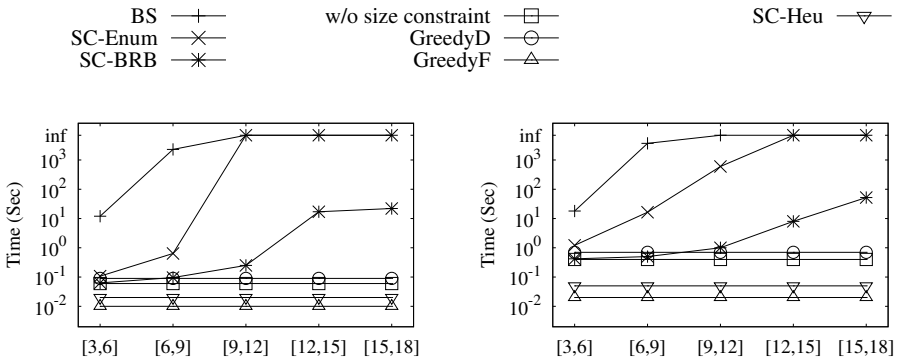
(c) GreedyF ($[\ell, h] = [15, 20]$) (d) SC-BRB ($[\ell, h] = [15, 20]$)

Figure 3.15: Case studies

varying $[\ell, h]$ is shown in Figure 3.16. We can see that SC-BRB significantly outperforms SC-Enum and BS due to our powerful pruning and bounding techniques. Moreover, BS due to lack of pruning and bounding techniques is also outperformed by SC-Enum. Thus, we exclude SC-Enum and BS from the remaining testings.

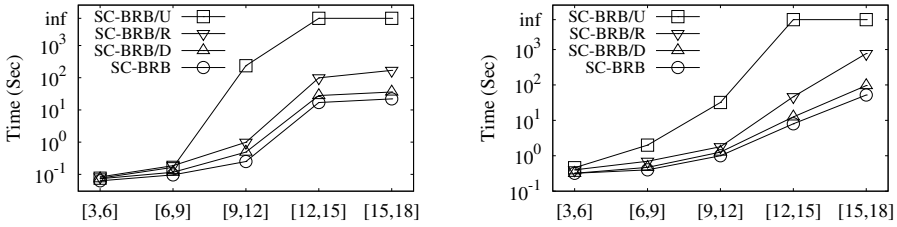
In Figure 3.16, we also include the running time of SC-Heu, GreedyD, GreedyF, and community search w/o size constraint. These heuristic algorithms run much faster than the exact algorithms. However, as demonstrated in Section 3.7.1, the result quality of GreedyD and GreedyF are not satisfactory. On the other hand, community search w/o size constraint will return extremely large communities (*e.g.*, almost the entire graph) (Chang and Qin, 2019). Consider the significant improvements on the result quality, it is cost-effective to apply SC-BRB. On the other hand, if time efficiency is critical, then our heuristic algorithm SC-Heu is recommended as it is superior than other alternatives.

Reducing, Upper Bounding, and Branching Technique. To evaluate our different techniques, we also implemented SC-BRB/R which is SC-BRB without our reduction rules, SC-BRB/U which is SC-BRB without our upper bounding techniques, and SC-BRB/D which is SC-BRB without our domination-based branching. Note that, SC-BRB/D still uses our connection score-based branching vertex selection. The running time of these algorithms on DBLP and Google by varying $[\ell, h]$ is shown in Figure 3.17. We can see that the running time of all algorithms increases when the size constraint increases, this is because the search space becomes larger. Nevertheless, SC-BRB consistently outperforms all other algorithms. We can also



(a) DBLP (b) Google

Figure 3.16: Against baseline algorithms (vary $[\ell, h]$)



(a) DBLP (b) Google

Figure 3.17: Evaluate our different techniques (vary $[\ell, h]$)

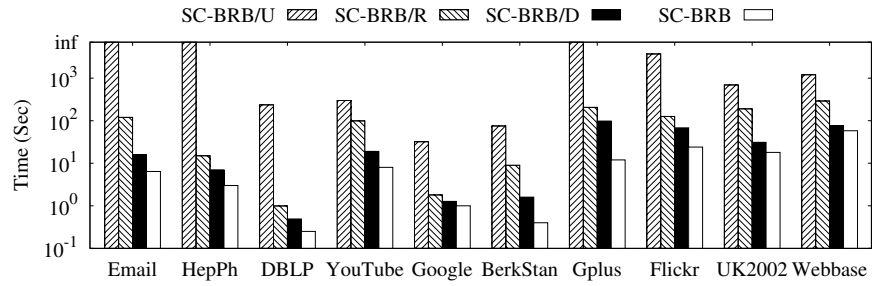
see that the running time increases, whenever any of the reducing, upper bounding, and branching techniques is removed. This confirms that all our techniques make a contribution to the performance of SC-BRB.

The running time of the algorithms on all real and synthetic graphs for $[\ell, h] = [9, 12]$ is shown in Figure 3.18. The trends are similar to Figure 3.17. For synthetic graphs, we also observe that the running time increases when the number of edges (correspondingly, density) of the graph increases. Nevertheless, the increase is sub-linear; recall that the number of edges of PL4 (resp. SBM4) is 10^3 times that of PL1 (resp. SBM1).

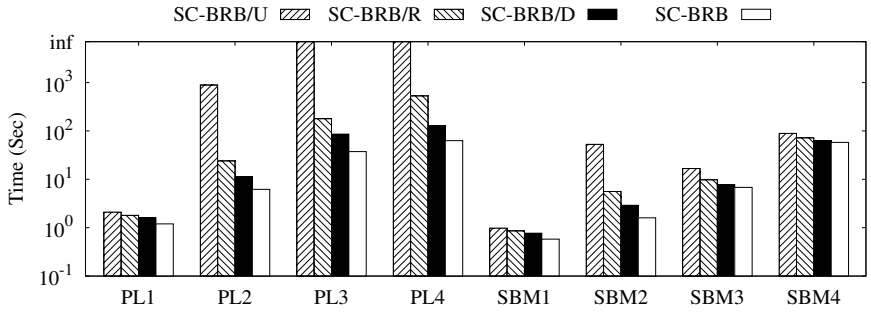
Evaluate Different Reduction Rules. We evaluate the effectiveness of the different reduction rules in reducing the running time. For simplicity, we use R1, R2 and R3 to represent our three reduction rules. We incrementally integrate these reduction rules into SC-BRB/R. The results in Figure 3.19 show that each of the three reduction rules reduces the running time.

Evaluate Different Upper Bounds. Figure 3.20 demonstrates the effectiveness of the three upper bounds, U_d, U_{nr} and U_{dc}, by incrementally adding them to SC-BRB/U. Each of the upper bounds reduces the running time. Thanks to the degree classification strategy, U_{dc} produces a tighter upper bound and is most powerful.

Evaluate Different Branching Vertex Selection. Figure 3.21 evaluates the effectiveness of the branching vertex selection. All our algorithms adopt *connection*

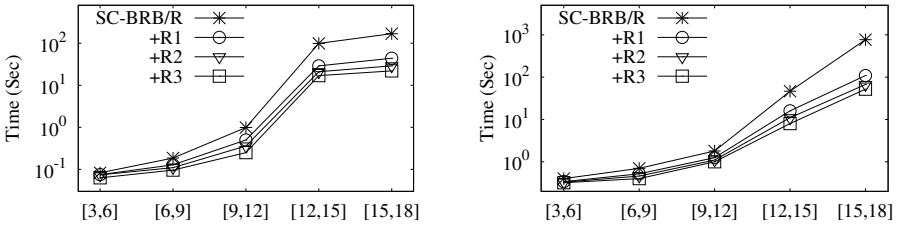


(a) Real graphs



(b) Synthetic graphs

Figure 3.18: Running time on all graphs ($[\ell, h] = [9, 12]$)



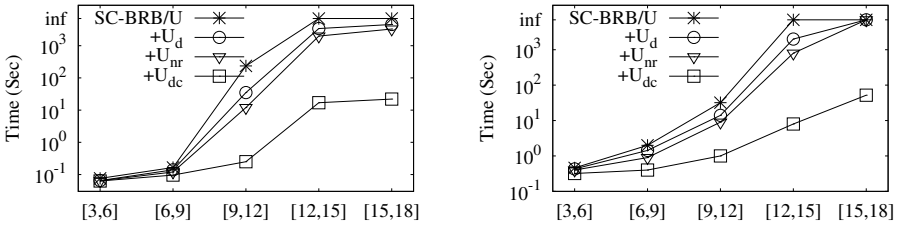
(a) DBLP (b) Google

Figure 3.19: Evaluate different reduction rules

score-based vertex selection, denoted $\delta(\cdot)$. Besides $\delta(\cdot)$, we also implement two other strategies: Random, and #Link, which selects the vertex of R that has the most number of links to C. The results demonstrate that the connection score-based strategy improves the efficiency significantly.

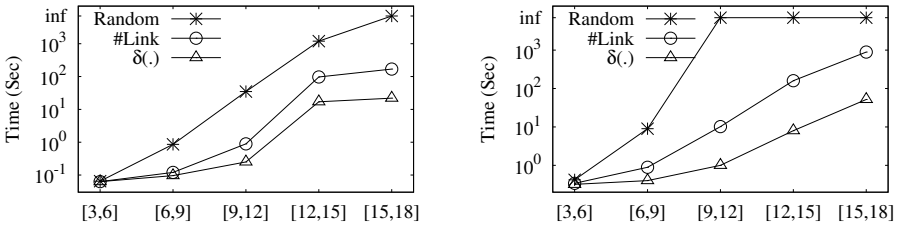
Evaluate Different Query-Range Sizes. We now evaluate the performance of SC-BRB under different query-range sizes. We fix $h = 15$ and increase ℓ from 3 to 15. Figure 3.22 shows that the running time increases along with the increasing of ℓ. This is because, when ℓ increases, it becomes harder to find a high-quality solution at early stages. Nevertheless, the influence of ℓ on the running time is not as strong as that of h (*e.g.*, as shown in Figure 3.16), as the running time of SC-BRB is upper bounded by n^h.

Evaluate Different Query Types. We evaluate the efficiency of SC-BRB for different query types: Dense, Sparse and Random. Random queries are generated as described at the beginning of this section. Dense query vertices are uniformly selected from the last 10% of the degeneracy ordering of all vertices (*i.e.*, from



(a) DBLP (b) Google

Figure 3.20: Evaluate different upper bounds



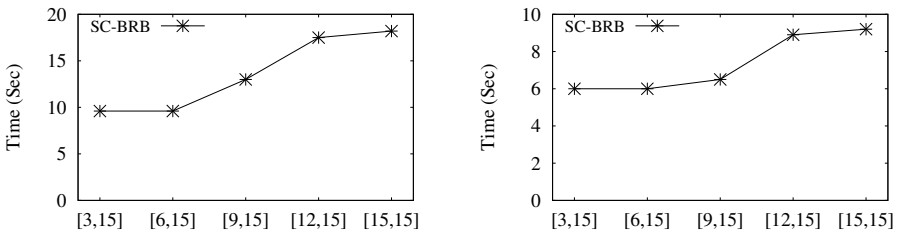
(a) DBLP (b) Google

Figure 3.21: Evaluate different branching vertex selection

dense parts of the graph), while Sparse query vertices are uniformly selected from the first 10% of the degeneracy ordering with core number at least 5 (*i.e.*, from sparse parts of the graph). The running time on all graphs for $[\ell, h] = [9, 12]$ is shown in Figure 3.23. We can see that Dense queries are the easiest to process, because we are more likely to find a high quality heuristic solution for Dense queries. Thus, our reduction rules and upper bounding techniques can significantly reduce the search space. Due to opposite reasons, Sparse queries are the hardest to process. Random queries are in between because they contain both Dense and Sparse queries.

3.7.3 SCS without Size Lower Bound

In this subsection, we evaluate SC-BRB against PSA for the problem of SCS without size lower bound. PSA was originally proposed for (approximately) computing the minimum k-core for a user-given integer k and query vertex q (Li et al., 2019), *i.e.*, the smallest one among all subgraphs that contain q and have minimum degree at



(a) DBLP (b) Google

Figure 3.22: Evaluate different query-range sizes

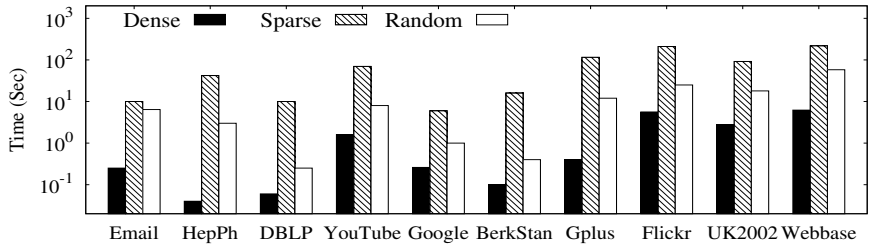
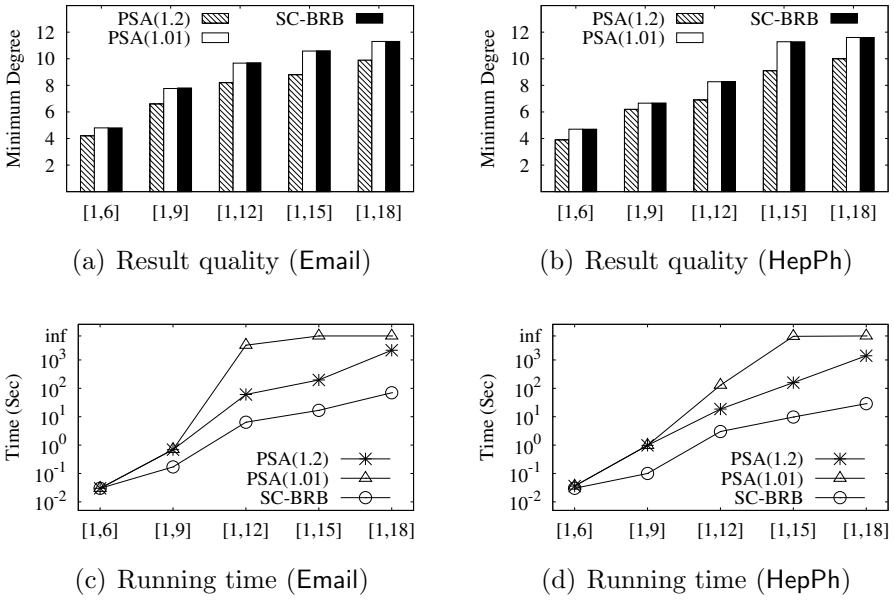


Figure 3.23: Different query types ($[\ell, h] = [9, 12]$)



(a) Result quality (Email)

(b) Result quality (HepPh)

(c) Running time (Email)

(d) Running time (HepPh)

Figure 3.24: Comparison with PSA by varying $[\ell, h]$

least k. In addition, PSA accepts a parameter $c \geq 1$ to strike a balance between the result quality and running time. Specifically, PSA returns a subgraph that is at most $c\times$ (the optimal size), instead of the optimal one. Intuitively, the smaller the value of c, the longer the running time. We adapt PSA for SCS without size lower bound, by computing an approximate minimum k-core for an iteratively increasing k until the size of the identified subgraph is larger than h. We report the penultimate k value as the result quality of PSA. The source code of PSA is obtained from the authors of (Li et al., 2019).

The running time and result quality of PSA(1.01), PSA(1.2) and SC-BRB on Email and HepPh by varying h are shown in Figure 3.24, where PSA(1.01) and PSA(1.2), respectively, represent PSA with $c = 1.01$ and $c = 1.2$. We can see that PSA(1.01) has the same result quality as SC-BRB due to the small value of $c = 1.01$, but is much slower than PSA(1.2) and SC-BRB and thus is not practical for large graphs. SC-BRB outperforms PSA(1.2) in terms of both result quality and running time.

Figure 3.25 shows the minimum degrees of the communities returned by PSA(1.2) and SC-BRB on all the graphs for $h = 10$. We can see that SC-BRB consistently outperforms PSA(1.2).

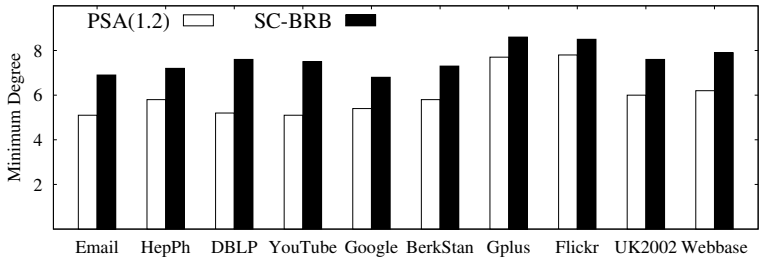


Figure 3.25: Comparison with PSA on all graphs ($h = 10$)

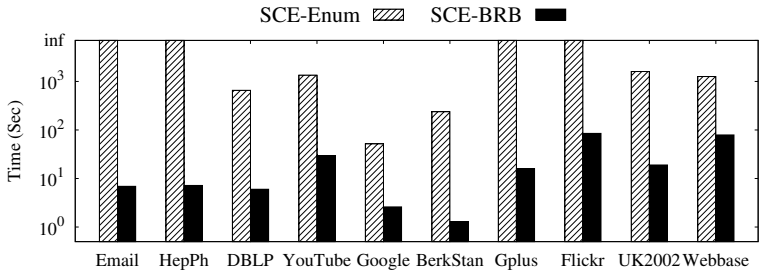


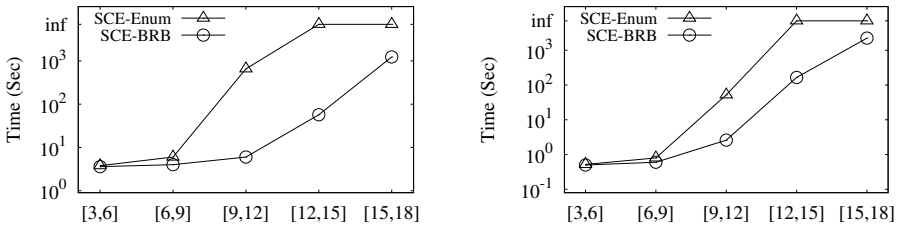
Figure 3.26: Running time on all graphs ($[\ell, h] = [9, 12]$)

3.7.4 Size-Bounded Community Enumeration Problem

In this subsection, we evaluate SCE-BRB against SCE-Enum for the size-bounded community enumeration problem. The running time of the algorithms on all graphs for $[\ell, h] = [9, 12]$ is shown in Figure 3.26. We can see that SCE-BRB consistently outperforms SCE-Enum. The speed up of SCE-BRB against SCE-Enum can be over 2 orders of magnitude. For example, on BerkStan, SCE-BRB enumerates all the size-bounded communities in 2 seconds while SCE-Enum uses 238 seconds to complete this task.

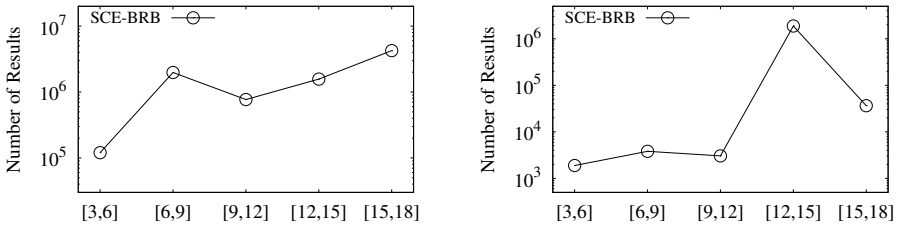
The running time of these algorithms on DBLP and Google by varying $[\ell, h]$ is shown in Figure 3.27. We can see that the running time of all algorithms increases when the size constraint increases, this is because the search space becomes larger. Nevertheless, SCE-BRB consistently outperforms SCE-Enum in all cases and the gap becomes larger when the size constraint increases. For example, on DBLP, SCE-BRB is slightly faster than SCE-Enum when the size constraint is $[3, 6]$ (i.e., 3.6 seconds v.s. 3.8 seconds). When the size constraint increases to $[9, 12]$, SCE-BRB is two orders of magnitude faster than SCE-Enum (i.e., 6 seconds v.s. 656 seconds). This experiment demonstrates the effectiveness of our optimization techniques in solving the size-bounded community enumeration problem.

Figure 3.28 reports the number of size-bounded communities on DBLP and Google by varying $[\ell, h]$. We can see that a large number of communities are detected in all cases. For example, on DBLP, the number of communities is $4,248,533$ when $[\ell, h]$ is $[15, 18]$. Thus, our algorithm can provide users with more choices. Another interesting phenomenon is that the number of results is not monotonically changed when the size constraint changes. For example, on DBLP, the number of communities increases from $120,110$ to $1,977,781$ when the size constraint increases from $[3, 6]$ to $[6, 9]$, while this number decreases to $768,208$ when we continue to increase the



(a) DBLP (b) Google

Figure 3.27: Evaluate our optimization techniques (vary $[\ell, h]$)



(a) DBLP (b) Google

Figure 3.28: Number of communities (vary $[\ell, h]$)

size constraint to $[9, 12]$.

3.8 Chapter Summary

In this chapter, we studied the problem of size-bounded community search which has both size lower bound and size upper bound, and aims to maximize the minimum degree of the returned subgraph. We proposed a branch-reduce-and-bound algorithm SC-BRB for solving the problem over large real graphs. SC-BRB outputs the optimal results. The efficiency of SC-BRB is due to our newly developed reducing techniques, upper bound techniques, and branching techniques. We extended our techniques to enumerate all optimal communities. Extensive experiments on large real graphs demonstrate the effectiveness and efficiency of our algorithms. As a possible direction of future work, it will be interesting to adapt our techniques to speed up the computation for the problems studied in (Li et al., 2019) and (Ma et al., 2019).

Chapter 4

Similar-Biclique Identification

In this chapter, we study the similar-biclique identification problem in bipartite graphs. The work is published in (Yao et al., 2022b). This chapter is organized as follows. Section 4.1 provides the introduction to this work. Section 4.2 provides the preliminaries including the definition of similar-biclique and the problem statement. Section 4.3 introduces a baseline algorithm and our MSBE algorithm. Section 4.4 presents our index structure, index-based algorithms and index construction algorithms. Section 4.5 introduces the index maintenance algorithms. Section 4.6 parallelizes our index construction algorithms. Section 4.7 reports experimental results. Section 4.8 concludes the chapter.

4.1 Introduction

Bipartite graphs have been widely used in real-world applications to model relationships between entities of different types, such as customer-product networks (Wang et al., 2006), author-paper networks (Ley, 2002) and user-event networks (EL BACHA and Zin, 2018). A bipartite graph is denoted by $G = (V_L, V_R, E)$, where the vertex set is partitioned into two disjoint subsets V_L and V_R which are referred to as the L-side and R-side vertices of the bipartite graph, respectively; each edge $e \in E$ can only connect vertices from different sides. Finding dense subgraphs in a bipartite graph is of great significance and encompasses many applications, such as community detection (Abidi et al., 2021; Lehmann et al., 2008), anomaly detection (Gangireddy et al., 2020; Sarıyüce and Pinar, 2018), and group recommendation (Lyu et al., 2020; Su and Khoshgoftaar, 2009).

One classic notion of dense bipartite subgraph is *biclique* (Peeters, 2003), which requires every pair of vertices from different sides of the subgraph to be connected by an edge. For example, for the bipartite graph in Figure 4.1 which represents researchers publishing in conference venues, the subgraph in the shadowed area is a biclique. In the literature, many algorithms have been proposed to enumerate all maximal bicliques (Abidi et al., 2021; Alexe et al., 2004; Eppstein, 1994; Li et al., 2007; Liu et al., 2006; Uno et al., 2004; Zhang et al., 2014) and to identify a biclique of the maximum size (Lyu et al., 2020). However, the biclique model has a fundamental limitation: vertices in a biclique are not necessarily similar to each other, despite that they share a set of common neighbors (i.e., vertices on the other

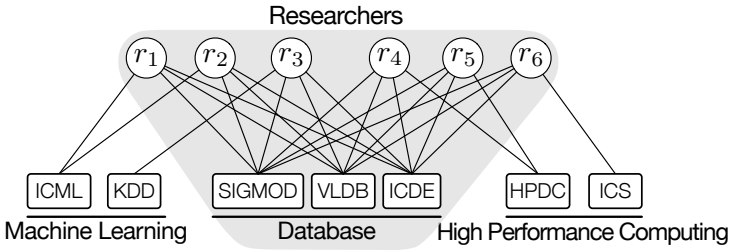


Figure 4.1: Example of researcher-venue bipartite graph

side of the biclique). Consider the six researchers in the biclique in Figure 4.1, all of them publish in database conferences. Besides, researchers r_1, r_2, r_3 also publish in machine learning (ML) conferences, while r_4, r_5, r_6 publish in high-performance computing (HPC) conferences. Thus, the two groups of researchers, $\{r_1, r_2, r_3\}$ and $\{r_4, r_5, r_6\}$, are likely from different backgrounds and communities, i.e., ML vs. HPC.

Motivated by this, we formulate the notion of *similar-biclique* by requiring all vertices from one side of the biclique to be similar to each other. Our empirical studies show that similar-bicliques can be detected much more efficiently than bicliques. Thus, identifying similar-bicliques is useful for the following applications.

- **Community detection.** Similar-biclique satisfies all the key requirements of community structure for bipartite graphs (Lehmann et al., 2008; Wang and Liu, 2018; Zhang and Ahn, 2015), and thus can be used to detect communities in interaction-type bipartite graphs such as user-rate-movie, customer-buy-product, and author-write-paper. Firstly, being a biclique, interactions between vertices from the two sides are intensive. Secondly, by enforcing the similarity constraint, users in a similar-biclique are similar to each other, e.g., having similar behaviours or interests.

- **Anomaly detection.** Similar-biclique can also be used for anomaly detection, which is a common task in e-commerce (Allahbakhsh et al., 2013; Ding et al., 2017; Lyu et al., 2020). Here, the transactions of customers purchasing products form a customer-product bipartite graph. To improve the ranking of certain products, fraudsters may create fake accounts to purchase the products, i.e., click farming (Dave et al., 2013). These fake accounts and the products they promote inevitably form a closely connected group, and meanwhile, these fake accounts will display a high level of synchronized behavior with each other (Jiang et al., 2014). Thus, suspicious groups (i.e., both the fraudulent accounts and the products they promote) can be captured by similar-bicliques.

Formally speaking, given a similarity threshold $0 < \varepsilon \leq 1$ and a size constraint $\tau \geq 0$, a vertex subset $C \subseteq V_L \cup V_R$ in a bipartite graph $G = (V_L, V_R, E)$ is a *similar-biclique* if (1) C is a biclique (i.e., $C_L \times C_R \subseteq E$), (2) all vertices of C_L are similar to each other (i.e., $\mathsf{sim}(u, v) \geq \varepsilon, \forall u, v \in C_L$), and (3) C satisfies the size constraint (i.e., $|C_L| \geq \tau$ and $|C_R| \geq \tau$). Here, C_L denotes $C \cap V_L$ and C_R denotes $C \cap V_R$; $\mathsf{sim}(u, v)$ measures the structural similarity between u and v, which is computed based on their neighbors $N(u)$ and $N(v)$ and will be formally defined

in Section 4.2; the size constraint τ is introduced to avoid generating too small or too skewed results. Note that, we only apply the similarity constraint to one side of the vertices (either V_L or V_R as they are interchangeable), since in applications we are usually only interested in the similarity between "users". Nevertheless, the general technical ideas presented in this chapter can also be applied to the variant of similar-biclique that imposes the similarity constraint on both sides of vertices.

We in this chapter aim to enumerate all *maximal* similar-bicliques in a bipartite graph. We prove that this problem is #P-complete. As each (maximal) similar-biclique is contained in a maximal biclique, we could first enumerate all maximal bicliques, then extract maximal similar-bicliques from maximal bicliques, and finally eliminate all similar-bicliques that are either duplicates or not maximal. However, this approach is inefficient, as enumerating all maximal bicliques by the state-of-the-art algorithm ooMBEA (Chen et al., 2022) is still time consuming for large graphs. Thus, we propose the MSBE algorithm to directly enumerate maximal similar-bicliques, without first enumerating maximal bicliques.

MSBE follows the general backtracking framework of the Bron-Kerbosch algorithm (Bron and Kerbosch, 1973) that enumerates all maximal *cliques* in a unipartite graph. Our observation is that once the set of L-side vertices C_L of a similar-biclique C is determined, its R-side vertices can be simply obtained as $C_R = \bigcap_{u \in C_L} N(u)$. Nevertheless, it is worth pointing out that we cannot ignore C_R during the enumeration process, since (1) the size of C_R will be used for pruning and (2) both C_L and C_R are needed for determining the maximality of the similar-biclique. MSBE iteratively builds up a partial solution $\langle C_L, C_R \rangle$, maintains a candidate set P_L that is used to grow C_L, and maintains an exclusive set Q_L that is used for checking the maximality of $\langle C_L, C_R \rangle$. In each recursion, a vertex $u \in P_L$ is added to C_L to grow the solution; after coming back from the recursion, u is moved from P_L to Q_L to avoid duplicates. To prune the search space, we propose the concept of *dominating*: $u \in P_L$ dominates $v \in P_L$ if $\mathsf{sim}(u, v) \geq \varepsilon$ and $N_{C_R}(u) \supseteq N_{C_R}(v)$, where $N_{C_R}(u) = N(u) \cap C_R$. We prove that if u dominates v, then we can prune the recursion of adding v to C_L when u is moved from P_L to Q_L. Furthermore, according to the definition, (1) each vertex u in a similar-biclique C must have at least τ neighbors in C (i.e., $|N_C(u)| \geq \tau$), and (2) each L-side vertex $u \in C_L$ must have at least $\tau - 1$ vertices that are similar to it; we call the vertices that are similar to u the *similar neighbors* of u, denoted $\Gamma(u)$. Thus, we propose to first prune all vertices that violate either of these two conditions, in a preprocess referred to as *vertex reduction*; our empirical studies show that a large portion of the input graph is pruned by vertex reduction.

We observe that a time-critical operation, in both vertex reduction and the recursion of backtracking, is computing $\Gamma(u)$ which would take $O(\sum_{v \in N(u)} |N(v)|)$ time, for a vertex $u \in V_L$. Note that, $\Gamma(u)$ is not stored in the graph representation, and it is also not affordable to store $\Gamma(u)$ (either in main memory or on disk) after it is computed as this would require a prohibitively large space. For example, it would take over 400GB on Bibsonomy, one of the graphs tested in our experiments. In view of this, we propose an offline-constructed index to speed up the computation of $\Gamma(u)$; note that, our index can be used to process maximal similar-biclique enumeration queries with different ε and τ values. This is based on the fact that for

any similarity threshold ε, $\Gamma(u)$ is always a subset of $\Phi_u = \bigcup_{v \in N(u)} N(v)$, the 2-hop structural neighbors of u. Thus, we propose to first compute the similarity between u and every vertex of Φ_u in an offline process, and then compress them into a few segments which are stored in the index. Specifically, each segment is represented by $\mathsf{seg} = \langle \mathsf{v}_{\min}, \mathsf{v}_{\max}, \mathsf{s}_{\max}, c \rangle$ where $\mathsf{v}_{\min} \leq \mathsf{v}_{\max}$ are two vertices of Φ_u, $\mathsf{s}_{\max}$ is the largest similarity between u and vertices of Φ_u that are in the range $[\mathsf{v}_{\min}, \mathsf{v}_{\max}]$, and c is the number of vertices of Φ_u that are in $[\mathsf{v}_{\min}, \mathsf{v}_{\max}]$; here, the comparison between vertices is based on their ids. To obtain $\Gamma(u)$, we go through each segment seg of Φ_u that satisfies $\mathsf{seg}.\mathsf{s}_{\max} \geq \varepsilon$, and compute the similarity between u and each $v \in [\mathsf{seg}.\mathsf{v}_{\min}, \mathsf{seg}.\mathsf{v}_{\max}]$; note that, segments with $\mathsf{s}_{\max} < \varepsilon$ are skipped. Furthermore, we also use the index to speed up vertex reduction by first pruning vertices based on upper bounds of $\Gamma(u)$, which can be efficiently obtained based on the index without computing similarities.

Our main contributions are summarized as follows:

- We formulate the concept of similar-biclique, which can be used to detect interesting dense subgraphs from a bipartite graph. To the best of our knowledge, this is the first work investigating structural similarity between vertices in dense bipartite subgraph mining.
- We develop a backtracking algorithm MSBE to enumerate all maximal similar-bicliques in a bipartite graph. Vertex reduction and optimization techniques are proposed.
- We design a novel index structure to facilitate the computation of similar neighbors and propose a two-phase algorithm for efficient vertex reduction based on the index.
- We propose effective and efficient index construction algorithms by investigating two different strategies.
- We propose index maintenance algorithms to handle dynamic graph updates.
- We parallelize our index construction algorithms by utilizing multiple CPU cores.
- We conduct extensive experiments on 17 real bipartite graphs to demonstrate the efficiency of our algorithms and the effectiveness of our similar-biclique model. Our algorithm is up to 6 orders of magnitude faster than ooMBEA.

4.2 Preliminaries

We consider an unweighted and undirected bipartite graph $G = (V_L, V_R, E)$, where V_L and V_R denote the two disjoint vertex sets (i.e., L-side vertices and R-side vertices) and $E \subseteq V_L \times V_R$ denotes the edge set. Without loss of generality, we assume that vertices of V_L take (integer) ids from $\{1, 2, \ldots, |V_L|\}$, and vertices of V_R take ids from $\{1 + |V_L|, 2 + |V_L|, \ldots, |V_R| + |V_L|\}$. For any vertex $v \in V_L$ (resp. $v \in V_R$), we say it is an L-side vertex (resp. R-side vertex). For any vertex subset $C \subseteq V_L \cup V_R$,

we use C_L and C_R to denote the subsets of vertices of C that are from V_L and V_R, respectively, i.e., $C_L = C \cap V_L$ and $C_R = C \cap V_R$. We call the set of neighbors of u in G, the *structural neighbors* of u, denoted $N_G(u) = \{v \mid (u, v) \in E\}$, and denote the *structural degree* of u by $d_G(u) = |N_G(u)|$. Besides structural neighbor and structural degree, we will also define similar neighbor and similar degree based on structural similarity.

Definition 3 (Structural Similarity)**.** Given two vertices u and v in G, their structural similarity is defined as $\mathsf{sim}(u, v) = \frac{|N_G(u) \cap N_G(v)|}{|N_G(u) \cup N_G(v)|}$.

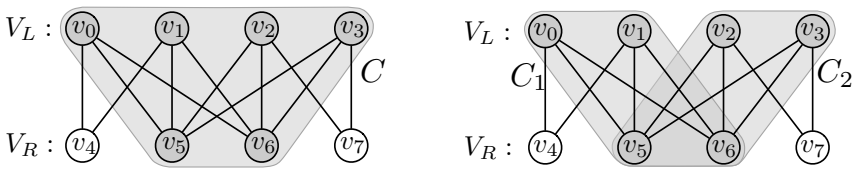
The structural similarity $\mathsf{sim}(u, v)$ is between 0 and 1. It measures the Jaccard similarity between the set of structural neighbors of u and that of v. Given a similarity threshold $\varepsilon > 0$, we say u and v are similar if $\mathsf{sim}(u, v) \geq \varepsilon$. The set of vertices that are similar to u is called the *similar neighbors* of u, denoted $\Gamma_{G,\varepsilon}(u)$, i.e., $\Gamma_{G,\varepsilon}(u) = \{v \in V_L \cup V_R \mid \mathsf{sim}(u, v) \geq \varepsilon\}$. Accordingly, denote $\delta_{G,\varepsilon}(u) = |\Gamma_{G,\varepsilon}(u)|$ the *similar degree* of u. Note that, as the structural similarity between vertices from different sides is always 0, similar neighbors only contain vertices from the same side. For presentation simplicity, we call structural similarity simply as *similarity*, and omit the subscripts G and/or ε from the notations.

Definition 4 (Similar-Biclique)**.** Given a bipartite graph $G = (V_L, V_R, E)$ and a similarity threshold $\varepsilon > 0$, a vertex subset $C \subseteq V_L \cup V_R$ is a similar-biclique if

- C is a *biclique*, i.e., $C_L \times C_R \subseteq E$, and
- all vertices from the L-side are similar to each other, i.e., $\mathsf{sim}(u, v) \geq \varepsilon, \forall u, v \in C_L$.

We also denote C as $\langle C_L, C_R \rangle$. A similar-biclique is *maximal* if it is not a subset of any larger similar-biclique.

Note that for presentation simplicity, the similarity constraint is assumed to be considered for the L-side vertices. To apply the similarity constraint for R-side vertices in applications, we can simply swap the roles of V_L and V_R in G.



(a) A maximal biclique (b) Two maximal similar-bicliques

Figure 4.2: Maximal biclique vs. maximal similar-biclique

Example 8. *Figure 4.2(a) shows a bipartite graph G with $V_L = \{v_0, \ldots, v_3\}$ and $V_R = \{v_4, \ldots, v_7\}$, in which the subgraph C in the gray area is a maximal biclique. However, v_1 and v_2 in the L-side of C are not similar to each other for $\varepsilon = 0.6$, since $\mathsf{sim}(v_1, v_2) = 0.5$. Subgraphs C_1 and C_2 as shown in Figure 4.2(b) are two maximal similar-bicliques, in each of which all vertices on the L-side are similar to each other.*

Problem 2 (Maximal Similar-Biclique Enumeration)**.** Given a bipartite graph $G = (V_L, V_R, E)$, a similarity threshold $\varepsilon > 0$ and a size constraint $\tau > 0$, we study the problem of enumerating all maximal similar-bicliques C in G that satisfy the size constraint τ (i.e., $|C_L| \geq \tau$ and $|C_R| \geq \tau$).

The size constraint τ is adopted here to avoid generating too small or too skewed similar-bicliques (i.e., with very few or no vertices in one side). For presentation simplicity, we assume the same size constraint τ for both sides. Note that, the techniques that we are going to present in this chapter can be straightforwardly extended to handle different size constraints on the two sides.

Theorem 3. *The problem of enumerating all maximal similar-bicliques is #P-complete.*

Proof. The #P-completeness of our problem directly follows from the facts that (1) the problem of enumerating all maximal bicliques is #P-complete (Kloster et al., 2019; Kuznetsov, 2001), and (2) it is a special case of our problem, i.e., by setting $\varepsilon = \frac{1}{|V_R|}$. Note that, for this small ε, two vertices of V_L are similar if and only if they have at least one common neighbors in V_R. Thus, every maximal similar-biclique is also a maximal biclique, and vice versa. □

Remark about Strctural Similarity. In Definition 3, we use the Jaccard similarity to measure the structural similarity since it has been widely used and shown great success in many graph analysis tasks, such as structural graph clustering (Chang et al., 2017; Tseng et al., 2021), link prediction (Liben-Nowell and Kleinberg, 2007; Martínez et al., 2016), and local graph sparsification (Satuluri et al., 2011). Nevertheless, other local-topology-based similarity measures, such as cosine similarity: $\frac{|N_G(u) \cap N_G(v)|}{\sqrt{d_G(u) \times d_G(v)}}$, hub promoted index: $\frac{|N_G(u) \cap N_G(v)|}{\min\{d_G(u), d_G(v)\}}$, can be easily plugged into our model and algorithms. We will point out the changes that need to be made to adopt these measures, when presenting our algorithms.

4.3 Our Algorithms

In this section, we propose an MSBE algorithm to enumerate all maximal similar-bicliques. Before that, we first in Section 4.3.1 present a baseline algorithm based on the existing maximal biclique enumeration algorithms.

4.3.1 A Baseline Algorithm

It is easy to observe that maximal similar-bicliques must be contained in maximal bicliques, since every similar-biclique is also a biclique. Thus, a naive approach is first enumerating all maximal bicliques by invoking one of the existing maximal biclique enumeration algorithms, and then post-process the detected maximal bicliques to obtain maximal similar-bicliques. Specifically, for each maximal biclique, we extract maximal similar-bicliques by imposing the similarity constraint on L-side vertices, and then eliminate all similar-bicliques that are either duplicates or non-maximal. We omit the details, since our empirical study in Section 4.7 shows that enumerating

Algorithm 7: MSBE$(G = (V_L, V_R, E), \varepsilon, \tau)$

```
1  for each u ∈ V_L ∪ V_R do del(u) = false;
2  VReduce(G, ε, τ, del(·));
3  for each u ∈ V_L s.t. del(u) = false do
4      C_L ← {u}; C_R ← {v ∈ N(u) | del(v) = false};
5      P_L ← ∅; Q_L ← ∅;
6      Obtain Γ(u);
7      for each v ∈ Γ(u) do
8          if v > u then P_L ← P_L ∪ {v} ;
9          else Q_L ← Q_L ∪ {v};
10     Enum(C_L, C_R, P_L, Q_L);

   Procedure Enum(C_L, C_R, P_L, Q_L)
11 if ∄u ∈ P_L ∪ Q_L s.t. N(u) ⊇ C_R then
12     if |C_L| ≥ τ and |C_R| ≥ τ then output ⟨C_L, C_R⟩ ;
13 for each u ∈ P_L do
14     C'_L ← C_L ∪ {u}; C'_R ← C_R ∩ N(u);
15     Obtain Γ(u);
16     P'_L ← P_L ∩ Γ(u); Q'_L ← Q_L ∩ Γ(u);
17     if |C'_L| + |P'_L| ≥ τ and |C'_R| ≥ τ then Enum(C'_L, C'_R, P'_L, Q'_L) ;
18     P_L ← P_L \ {u}; Q_L ← Q_L ∪ {u};
```

all maximal bicliques by the state-of-the-art algorithm ooMBEA (Chen et al., 2022) is already time consuming for large graphs.

4.3.2 Our MSBE Algorithm

According to the definition of similar-biclique, if we build a similarity graph G_s for V_L where two vertices of V_L are connected by an edge if their similarity is at least ε, then for every (maximal) similar-biclique C, its L-side vertices C_L form a clique in G_s. Moreover, once the L-side vertices C_L of a maximal similar-biclique are determined, the R-side vertices C_R can be easily obtained as the set of common neighbors of C_L, i.e., $C_R = \bigcap_{u \in C_L} N(u)$. In view of this, we propose to adopt the general backtracking framework of the Bron-Kerbosch algorithm (Bron and Kerbosch, 1973) to enumerate all maximal similar-bicliques. However, there are two issues. (1) The similarity graph G_s is not available in the input. (2) The set of L-side vertices C_L of a maximal similar-biclique C is not necessarily a *maximal* clique in G_s, though C_L is a clique in G_s. This is because, a too large C_L may result in a too small C_R, violating the size constraint τ on C_R.

We propose techniques to address the above issues, and the pseudocode of our algorithm is shown in Algorithm 7, denoted MSBE. We first prune unpromising vertices by invoking VReduce (Lines 1–2), which will be introduced shortly in Algorithm 8; a vertex u is pruned if $\mathsf{del}(u) = \textbf{true}$. For each remaining vertex $u \in V_L$ with $\mathsf{del}(u) = \textbf{false}$, we enumerate all maximal similar-bicliques containing u (Lines 3–10). To do so, we iteratively grow a partial solution $\langle C_L, C_R \rangle$ where C_L is initialized as $\{u\}$ (Line 4). Besides C_L and C_R, we also maintain P_L and Q_L, in a similar fashion to the Bron-Kerbosch algorithm (Bron and Kerbosch, 1973). Specifically, P_L is a set of candidate vertices that is used to grow C_L, and Q_L is a set of previously

Algorithm 8: VReduce($G = (V_L, V_R, E), \varepsilon, \tau, \mathsf{del}(\cdot)$)

```
1 for each u ∈ V_L ∪ V_R do d(u) ← |N(u)|;
2 for each u ∈ V_L do Obtain Γ(u) and set δ(u) ← |Γ(u)|;
3 while (∃u ∈ V_L ∪ V_R s.t. del(u) = false and d(u) < τ) or (∃u ∈ V_L s.t. del(u) = false
    and δ(u) < τ − 1) do
4     for each v ∈ N(u) do d(v) ← d(v) − 1;
5     if u ∈ V_L then
6         Obtain Γ(u);
7         for each v ∈ Γ(u) do δ(v) ← δ(v) − 1;
8     del(u) = true;
```

considered candidate vertices that is used for checking the maximality of $\langle C_L, C_R \rangle$; note that, each vertex of $P_L \cup Q_L$ must be similar to all vertices of C_L. P_L and Q_L are initialized at Lines 5–9, after which we invoke the procedure Enum for enumeration (Line 10); to avoid duplicates, the similar neighbors of u with larger vertex id than u are put in P_L, and those with smaller vertex id are put in Q_L.

In Enum, if the current similar-biclique $\langle C_L, C_R \rangle$ is maximal, we report it (Lines 11–12); note that $\langle C_L, C_R \rangle$ is maximal if and only if there is no vertex $u \in P_L \cup Q_L$ such that $N(u) \supseteq C_R$. Next, Enum iteratively adds a vertex of P_L to C_L, updates the corresponding C_R, P_L and Q_L, and recursively invokes itself to enumerate more similar-bicliques (Lines 14–17). After processing $u \in P_L$, we remove u from P_L and add it to Q_L (Line 18).

Vertex Reduction. As a similar-biclique needs to have at least τ vertices on each side, each vertex u in a similar-biclique C must have at least τ structural neighbors in C (i.e., $|N_C(u)| \geq \tau$). Furthermore, as all L-side vertices in a similar-biclique C are similar to each other, each L-side vertex u must have at least $\tau - 1$ similar neighbors in C (i.e., $|\Gamma_C(u)| \geq \tau - 1$). As a result, we can exclude all vertices that violate either of these two conditions from being considered in the enumeration procedure Enum, i.e., mark them as deleted; we call this process as *vertex reduction.*

We propose an algorithm VReduce to conduct vertex reduction, whose pseudocode is shown in Algorithm 8. Firstly, we obtain the structural degree $d(u)$ for each vertex $u \in V_L \cup V_R$ (Line 1), and obtain the similar degree $\delta(u)$ for each L-side vertex $u \in V_L$ (Line 2). Then, as long as there is a non-deleted vertex u with $d(u) < \tau$ or a non-deleted L-side vertex u with $\delta(u) < \tau - 1$, we mark u as deleted (Line 8), decrease the structure degree of u's structural neighbors by 1 (Line 4), and decrease the similar degree of u's similar neighbors by 1 if u is an L-side vertex (Lines 5–7).

Compute Similar Neighbors $\Gamma(u)$. One fundamental operation in both Algorithm 7 and Algorithm 8 is computing $\Gamma(u)$ for an L-side vertex u; note that $\Gamma(u)$ is not stored with the graph G. A straightforward method to collect $\Gamma(u)$ is computing $\mathsf{sim}(u, v)$ for each vertex $v \in V_L$. The time complexity would be $O(|E|)$, as it needs to visit every edge of G. This is inefficient, by noting that Algorithm 7 and Algorithm 8 need to compute $\Gamma(u)$ for many vertices u and for multiple times.

We propose a more efficient algorithm in Algorithm 9, denoted SimNei. Instead of blindly computing $\mathsf{sim}(u, v)$ for each $v \in V_L$, we only compute $\mathsf{sim}(u, v)$ for those v with $\mathsf{sim}(u, v) > 0$. Our main idea is to first compute the number of common

Algorithm 9: SimNei$(G = (V_L, V_R, E), u, \varepsilon, \mathsf{del}(\cdot))$

Output: $\Gamma(u)$

1 $\Gamma(u) \leftarrow \emptyset$;
2 **for each** $v \in V_L$ **do** $c(v) \leftarrow 0$;
3 **for each** $v \in N(u)$ **do**
4 **for each** $w \in N(v)$ **and** $w \neq u$ **do** $c(w) \leftarrow c(w) + 1$;
5 **for each** $v \in V_L$ *s.t.* $c(v) \neq 0$ **and** $\mathsf{del}(v) =$ **false do**
6 **if** $\frac{c(v)}{d(u)+d(v)-c(v)} \geq \varepsilon$ **then** $\Gamma(u) \leftarrow \Gamma(u) \cup \{v\}$;
7 **return** $\Gamma(u)$;

neighbors $c(v)$ between u and v for each 2-hop structural neighbor v of u (Lines 3–4). Then, $\mathsf{sim}(u, v)$ can be calculated as $\frac{c(v)}{d(u)+d(v)-c(v)}$ (Line 6).[1] Note that, in our implementation, to make the time complexity of SimNei to be independent of $|V_L|$ which may be large, we only initialize $c(\cdot)$ once at the beginning of the entire algorithm execution (e.g., in MSBE), and after using $c(\cdot)$ at Line 4–6 of Algorithm 9 we reset those updated $c(\cdot)$ to be 0. In addition, we also collect at Line 4 the set of vertices with non-zero $c(\cdot)$ into a set S, such that Line 5 as well as the resetting of $c(\cdot)$ can be conducted in $O(|S|)$ time. As a result, the time complexity of SimNei is $O(\sum_{v \in N(u)} d(v))$, which is lower than $O(|E|)$.

Optimizations for Enum. We further propose two optimization techniques to speed up the Enum procedure. Recall that, an instance of Enum is represented by (C_L, C_R, P_L, Q_L) where $C_R = \bigcap_{u \in C_L} N(u)$ and $|C_R| \geq \tau$,[2] and aims to enumerate all maximal similar-bicliques C^* satisfying $C_L \subset C_L^* \subseteq C_L \cup P_L$. Firstly, an enumeration instance can be terminated once we know that it will not generate any maximal similar-biclique. That is, by including any subset of vertices of P_L into C_L, the resulting similar-biclique is not maximal. This is formulated in the lemma below.

Lemma 7 (Early Termination)**.** *If there exists a vertex $u \in Q_L$ such that u is similar to all vertices of P_L and $N(u) \supseteq C_R$, then there is no maximal similar-biclique C^* with $C_L \subset C_L^* \subseteq C_L \cup P_L$ and thus we can terminate this enumeration instance.*

Proof. Suppose there is such a maximal similar-biclique C^* with $C_L \subset C_L^* \subseteq C_L \cup P_L$. Then, we must have $C_R^* \subseteq C_R$ and thus $N(u) \supseteq C_R \supseteq C_R^*$. Since $u \in Q_L$ is similar to all vertices of P_L (and also note that all vertices of Q_L, including u, are similar to all vertices of C_L according to the construction of Q_L), u is similar to all vertices of C_L^*. Consequently, $C^* \cup \{u\}$ is also a similar-biclique, contradicting that C^* is a maximal similar-biclique. □

Secondly, we can reduce the number of instances generated at Line 17 of Algorithm 7, based on the concept of dominating set.

Definition 5 (Dominating Set)**.** Given an instance (C_L, C_R, P_L, Q_L) of Enum, for two distinct vertices $u, v \in P_L \cup Q_L$, we say that u dominates v if $\mathsf{sim}(u, v) \geq \varepsilon$

[1] The formula should be changed to $\frac{c(v)}{\sqrt{d(u) \times d(v)}}$ for cosine similarity, and to $\frac{c(v)}{\min\{d(u), d(v)\}}$ for hub promoted index.

[2] To be more precise, we should exclude from C_R all vertices that are marked as deleted.

and $N_{C_R}(u) \supseteq N_{C_R}(v)$, where $N_{C_R}(u) = N(u) \cap C_R$. The dominating set of u, denoted $\mathsf{DomSet}(u)$, is the subset of vertices of P_L that are dominated by u, i.e., $\mathsf{DomSet}(u) = \{v \in P_L \mid \mathsf{sim}(u, v) \geq \varepsilon \wedge N_{C_R}(u) \supseteq N_{C_R}(v)\}$.

Note that, a vertex does not dominate itself.

Lemma 8. *Given an instance (C_L, C_R, P_L, Q_L) of* Enum *and a vertex $u^* \in P_L \cup Q_L$, any maximal similar-biclique C^* with $C_L \subset C_L^* \subseteq C_L \cup P_L$ must contain a vertex of $P_L \backslash \mathsf{DomSet}(u^*)$.*

Proof. Suppose there is such a maximal similar-biclique C^* with $C_L \subset C_L^* \subseteq C_L \cup P_L$ that contains no vertex of $P_L \backslash \mathsf{DomSet}(u^*)$. That is, $C_L^* \backslash C_L \subseteq \mathsf{DomSet}(u^*)$. Then, it is easy to verify that $C^* \cup \{u^*\}$ is also a similar-biclique, contradicting the maximality of C^*. □

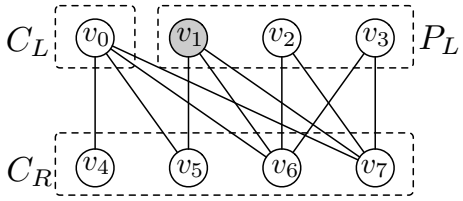


Figure 4.3: Example of domination

Example 9. *Consider the instance in Figure 4.3 where $Q_L = \emptyset$. For $\varepsilon = 0.6$, $v_1 \in P_L$ is similar to both v_2 and v_3. Moreover, we can see that $N_{C_R}(v_1) \supseteq N_{C_R}(v_2)$ and $N_{C_R}(v_1) \supseteq N_{C_R}(v_3)$. Thus, $\mathsf{DomSet}(v_1) = \{v_2, v_3\}$, and we know that every maximal similar-biclique C^* with $C_L \subset C_L^* \subseteq C_L \cup P_L$ must contain v_1 since $P_L \backslash \mathsf{DomSet}(v_1) = \{v_1\}$.*

To apply Lemma 8, we revise Line 13 of Algorithm 7 as follows: we first choose a vertex u^* from $P_L \cup Q_L$ before the "for loop", and then replace "$u \in P_L$" with "$u \in P_L \backslash \mathsf{DomSet}(u^*)$" in the "for loop" statement. This means that we do not generate enumeration instances, at Line 17 of Algorithm 7, for vertices $u \in \mathsf{DomSet}(u^*)$. To maximize the benefit of Lemma 8, u^* is chosen as the one that minimizes $|P_L \backslash \mathsf{DomSet}(u^*)|$ among all vertices of $P_L \cup Q_L$.

Theorem 4. *The time complexity of Algorithm 7 is $\mathcal{O}(|V_L| \cdot |E| \cdot 2^{|V_L|})$.*

Proof. Firstly, we prove that the time complexity of VReduce (*i.e.*, Algorithm 8) is $\mathcal{O}(\sum_{v \in V_R} (d(v))^2)$. In Algorithm 8, Line 1 runs in $\mathcal{O}(|E|)$ time. Line 2 runs in $\mathcal{O}(\sum_{v \in V_R} (d(v))^2)$ time, since computing $\Gamma(u)$ by Algorithm 9 takes $\mathcal{O}(\sum_{v \in N(u)} d(v))$ as discussed above; note that $\sum_{u \in V_L} \sum_{v \in N(u)} d(v) = \sum_{(u,v) \in E} d(v) = \sum_{v \in V_R} \sum_{u \in N(v)} d(v) = \sum_{v \in V_R} (d(v))^2$. Since each vertex $u \in V_L \cup V_R$ is removed at most once at Lines 4–8, the total cost of running Line 4 for all deleted vertices is $\mathcal{O}(|E|)$, and the total cost of running Lines 6–7 for all deleted vertices is $\mathcal{O}(\sum_{v \in V_R} (d(v))^2)$, the same as that of Line 2. In addition, we use a queue to store the vertices that should be deleted (i.e., satisfying the conditions at Line 3) such that finding a vertex at Line 3 takes constant time. Thus, the time complexity of VReduce is $\mathcal{O}(\sum_{v \in V_R} (d(v))^2)$.

Now, we prove that the time complexity of MSBE (Algorithm 7) is $\mathcal{O}(|V_L| \cdot |E| \cdot 2^{|V_L|})$. Firstly, it is easy to see that the time complexity of Algorithm 7 excluding Line 10 (i.e., the recursion) is $\mathcal{O}(\sum_{v \in V_R}(d(v))^2)$, by following the analysis of the time complexity of VReduce. Secondly, invoking Enum with input (C_L, C_R, P_L, Q_L) takes time $\mathcal{O}(|E| \cdot 2^{|P_L|})$, since the recursion builds a complete binary search tree with each instance (C_L, C_R, P_L, Q_L) has two children: one including u into C_L, one excluding u from C_L. The time for generating the first child (Lines 14–17) is $\mathcal{O}(\sum_{v \in N(u)} d(v)) \subseteq \mathcal{O}(|E|)$, and the time for generating the second child is $\mathcal{O}(1)$ (Line 18). In addition, the total number of leaf instances is 2^{P_L}. Thus, each invocation to Enum at Line 10 of Algorithm 7 takes time $\mathcal{O}(|E| \cdot 2^{|P_L|}) \subseteq \mathcal{O}(|E| \cdot 2^{|V_L|})$, and the total time complexity follows. □

Discussions. MSBE is different from both maximal clique enumeration algorithms for unipartite graphs and maximal biclique enumeration algoirthms for bipartie graphs, as follows. Firstly, MSBE needs to compute the similar neighbors for L-side vertices, which are not required by any of the existing algorithms. Secondly, compared with maximal clique enumeration algorithms, MSBE needs to consider common structural neighbors C_R of C_L, in addition to common similar neighbors. Thirdly, compared with the state-of-the-art algorithm ooMBEA for maximal biclique enumeration, MSBE needs to maintain the set Q_L for checking maximality of similar-bicliques. Forthly, our optimization techniques for Enum are also different.

In MSBE, we need to obtain the similar neighbors $\Gamma(\cdot)$ of an L-side vertex multiple times, e.g., at Lines 6 and 15 of Algorithm 7 and Lines 2 and 6 of Algorithm 8. We can either invoke SimNei to compute $\Gamma(u)$ every time when it is needed, or store $\Gamma(u)$ in main memory after it is computed for the first time and then directly retrieve it for all subsequent requests. We use MSBE to denote the algorithm that uses the first strategy, and mat-MSBE the algorithm that uses the second strategy (here, mat stands for materialization).

4.4 Speeding Up Similar Neighbor Computation and Vertex Reduction

MSBE has the disadvantage of repeatedly computing the similar neighbors from scratch which is time consuming, while mat-MSBE may demand an extremely large main memory to store the similar neighbors. For example, it would take more than 400GB main memory for the graph bibsonomy used in our experiments even for a moderate $\varepsilon = 0.5$. In this section, we propose an offline-constructed index to speed up the computation of $\Gamma(u)$ as well as vertex reduction. We give an overview of the index structure in Section 4.4.1, present our index-based algorithms in Section 4.4.2, and discuss index construction in Section 4.4.3.

4.4.1 Overview of Index Structure

Let Φ_u be the set of 2-hop structural neighbors of u, i.e., $\Phi_u = \bigcup_{v \in N(u)} N(v)$. Firstly, we have the following lemma.

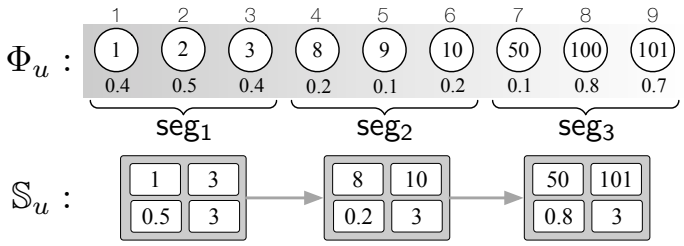


Figure 4.4: Overview of index structure

Lemma 9. *For any similarity threshold $\varepsilon > 0$, the set of similar neighbors of u is a subset of Φ_u, i.e., $\Gamma_\varepsilon(u) \subseteq \Phi_u$.*

Proof. The correctness of the lemma directly follows from the fact that any vertex $v \notin \Phi_u \cup \{u\}$ has no common neighbor with u and thus $\mathsf{sim}(u, v) = 0$. □

Based on Lemma 9, one possible indexing strategy is pre-computing and storing $\mathsf{sim}(u, v)$ for each $u \in V_L \cup V_R$ and each $v \in \Phi_u$.[3] However, the space complexity of this strategy would be $O(|V_L|^2 + |V_R|^2)$, which is prohibitively high even for moderate-sized graphs since the space requirement is essentially the same as the case of mat-MSBE when ε is very small. For example, even for a moderate-sized graph with 10^6 vertices, the storage space would be over 2TB.

Instead of storing Φ_u and the structural similarities in their raw format, we summarize them into a few segments.

Definition 6 (Segment)**.** A segment, denoted seg, of Φ_u is a four-tuple $\langle \mathsf{v}_{\min}, \mathsf{v}_{\max}, \mathsf{s}_{\max}, c \rangle$, where $\mathsf{v}_{\min} \leq \mathsf{v}_{\max}$ are two vertices of Φ_u, $\mathsf{s}_{\max} = \max_{v \in \Phi_u : \mathsf{v}_{\min} \leq v \leq \mathsf{v}_{\max}} \mathsf{sim}(u, v)$, and $c = |\{v \in \Phi_u \mid \mathsf{v}_{\min} \leq v \leq \mathsf{v}_{\max}\}|$. Here, vertex comparison is based on vertex id.

Given a segment $\mathsf{seg} = \langle \mathsf{v}_{\min}, \mathsf{v}_{\max}, \mathsf{s}_{\max}, c \rangle$ of Φ_u, we use $V(\mathsf{seg})$ to denote $\{v \in \Phi_u \mid \mathsf{v}_{\min} \leq v \leq \mathsf{v}_{\max}\}$. It is immediate from the definition that $c = |V(\mathsf{seg})|$ and

- $\mathsf{v}_{\min}$ (resp. $\mathsf{v}_{\max}$) is the smallest (resp. largest) vertex id in $V(\mathsf{seg})$;
- $\mathsf{s}_{\max}$ is the largest similarity between u and a vertex of $V(\mathsf{seg})$, and thus $\mathsf{s}_{\max}$ provides an upper bound of $\mathsf{sim}(u, v)$ for all $v \in V(\mathsf{seg})$.

Thus, we say that seg *covers* vertices $V(\mathsf{seg})$. *A set of segments* $\mathbb{S}_u = \{\mathsf{seg}_1, \ldots, \mathsf{seg}_k\}$ *covers* Φ_u *if* $\bigcup_{\mathsf{seg} \in \mathbb{S}_u} V(\mathsf{seg}) = \Phi_u$. In this chapter, we only consider disjoint segments, i.e., $V(\mathsf{seg}_i) \cap V(\mathsf{seg}_j) = \emptyset$ for $i \neq j$. Our index structure, denoted $\mathcal{I}$, covers Φ_u by a set of segments, for all $u \in V_L \cup V_R$. That is, $\mathcal{I}$ consists of $\mathbb{S}_u$ such that $\mathbb{S}_u$ covers Φ_u, for all $u \in V_L \cup V_R$.

Example 10. *Figure 4.4 shows the 2-hop structural neighbors Φ_u of u, which are sorted in increasing order regarding vertex id. The decimal below each vertex is the similarity w.r.t. u. Φ_u is covered by three segments seg_1, seg_2, seg_3. Take seg_1 as an example, the two numbers in the first row (i.e., 1 and 3) represent $\mathsf{v}_{\min}$ and $\mathsf{v}_{\max}$, and the two numbers in the second row (i.e., 0.5 and 3) represent $\mathsf{s}_{\max}$ and c.*

[3]Note that, we also need to index Φ_u for $u \in V_R$, since in practice the similarity constraint can be put on either L-side or R-side vertices.

Algorithm 10: indexedSN$(u, \varepsilon, G, \mathcal{I}, \mathsf{del}(\cdot))$

```
1 Γ(u) ← ∅;
2 for each seg ∈ 𝕊_u s.t. seg.s_max ≥ ε do
3     for each v ∈ [seg.v_min, seg.v_max] do
4         if del(v) = false and v ≠ u then
5             if ub(u, v) ≥ ε and sim(u, v) ≥ ε then
6                 Γ(u) ← Γ(u) ∪ {v};
7 return Γ(u);
```

4.4.2 Index-based Algorithms

In this subsection, we present index-based algorithms for similar neighbor computation and for vertex reduction.

Index-based Similar Neighbor Computation. The pseudocode of using the index $\mathcal{I}$ to efficiently obtain the similar neighbors $\Gamma(u)$ for a vertex u is shown in Algorithm 10, denoted indexedSN. We go through each segment $\mathsf{seg} \in \mathbb{S}_u$ with $\mathsf{seg.s}_{\max} \geq \varepsilon$ (Line 2), and compute $\mathsf{sim}(u, v)$ for each $v \in [\mathsf{seg.v}_{\min}, \mathsf{seg.v}_{\max}]$ (Line 3); recall that (1) $\mathsf{seg.s}_{\max}$ upper bounds $\mathsf{sim}(u, v)$ for each $v \in V(\mathsf{seg})$, and (2) $V(\mathsf{seg})$ is not stored in the index structure $\mathcal{I}$. As computing $\mathsf{sim}(u, v)$ needs to intersect two sets $N(u)$ and $N(v)$ which is costly, we propose to first apply a filtering for the pair u and v based on an upper bound $\mathsf{ub}(u, v)$ of $\mathsf{sim}(u, v)$ (Line 5); if $\mathsf{ub}(u, v) < \varepsilon$, then we have $\mathsf{sim}(u, v) \leq \mathsf{ub}(u, v) < \varepsilon$ and thus $v \notin \Gamma(u)$. For the similarity in Definition 3, it is easy to verify that $\mathsf{sim}(u, v) \leq \frac{\min\{d(u), d(v)\}}{\max\{d(u), d(v)\}}$; we set this as $\mathsf{ub}(u, v)$, which can be calculated in constant time.[4] indexedSN is expected to run faster than SimNei (Algorithm 9) as the former can skip an entire segment if its $\mathsf{s}_{\max}$ is smaller than ε.

Index-based Two-Phase Vertex Reduction. Based on indexedSN, we can speed up VReduce (Algorithm 8) by invoking indexedSN to compute $\Gamma(u)$. However, this is still inefficient, as VReduce needs to compute $\Gamma(u)$ for all $u \in V_L$ (see Line 2 of Algorithm 8). We propose to utilize the index $\mathcal{I}$ to first obtain an upper bound of the similar degree for vertex reduction, as proved in the lemma below.

Lemma 10 (Upper Bound of Similar Degree)**.** *Let $\mathbb{S}_u$ be the set of segments that cover Φ_u. Then, the similar degree $\delta_\varepsilon(u)$ of u is upper bounded by $\sum_{\mathsf{seg} \in \mathbb{S}_u : \mathsf{seg.s}_{\max} \geq \varepsilon} \mathsf{seg}.c$.*

Proof. This lemma directly follows from the fact that $\mathsf{sim}(u, v) < \varepsilon$ for all $v \in \bigcup_{\mathsf{seg} \in \mathbb{S}_u : \mathsf{seg.s}_{\max} < \varepsilon} V(\mathsf{seg})$. □

Consider the part of the index in Figure 4.4 and suppose $\varepsilon = 0.4$. By scanning $\mathbb{S}_u$, we obtain an upper bound of u's similar degree as 6, i.e., $\mathsf{seg}_1.c + \mathsf{seg}_3.c = 6$; seg_2 is omitted since its $\mathsf{s}_{\max}$ is only 0.2.

Furthermore, we also observe that the structural degree can be obtained efficiently. Thus, we propose a two-phase approach for vertex reduction, which first

[4] The upper bound for cosine similarity is $\frac{\min\{d(u), d(v)\}}{\sqrt{d(u) \times d(v)}}$, while the upper bound for hub promoted index is 1 and thus not useful.

Algorithm 11: indexedVR$(G, \mathcal{I}, \varepsilon, \tau, \mathsf{del}(\cdot))$

```
   /* Phase-I: vertex reduction based on structural degree and upper bound of similar degree */
1  for each u ∈ V_L ∪ V_R do d(u) ← |N(u)|;
2  for each u ∈ V_L do δ̄(u) ← Σ_{seg∈S_u : seg.s_max≥ε} seg.c;
3  while (∃u ∈ V_L ∪ V_R s.t. del(u) = false and d(u) < τ) or (∃u ∈ V_L s.t. del(u) = false
     and δ̄(u) < τ − 1) do
4      for each v ∈ N(u) do d(v) ← d(v) − 1;
5      del(u) ← true;
   /* Phase-II: vertex reduction based on structural degree and similar degree */
6  for each u ∈ V_L ∪ V_R do del2(u) ← del(u);
7  for each u ∈ V_L s.t. del(u) = false do
8      (δ_p(u), idx(u)) ← progressiveSN(u, ε, G, I, del2(·), τ − 1, 1);
9  while (∃u ∈ V_L ∪ V_R s.t. del(u) = false and d(u) < τ) or (∃u ∈ V_L s.t. del(u) = false
     and δ_p(u) < τ − 1) do
10     for each v ∈ N(u) do d(v) ← d(v) − 1;
11     if u ∈ V_L then
12         Γ(u) ← indexedSN(u, ε, G, I, del(·));
13         for each v ∈ Γ(u) do
14             δ_p(v) ← δ_p(v) − 1;
15             if δ_p(v) = τ − 2 and d(v) ≥ τ then
16                 (r, idx(v)) ← progressiveSN(v, ε, G, I, del2(·), 1, idx(v));
17                 δ_p(v) ← δ_p(v) + r;
18     del(u) ← true;
   Procedure progressiveSN(u, ε, G, I, del2(·), t, b)
   /* Let S_u be {seg_1, seg_2, ..., seg_{|S_u|}} */
19 r ← 0;
20 for each i ∈ {b, b + 1, ..., |S_u|} s.t. seg_i.s_max ≥ ε do
21     for each v ∈ [seg_i.v_min, seg_i.v_max] do
22         if del2(v) = false and v ≠ u then
23             if ub(u, v) ≥ ε and sim(u, v) ≥ ε then
24                 r ← r + 1;
25     if r ≥ t then return (r, i + 1);
26 return (r, |S_u| + 1);
```

conducts vertex reduction by using structural degree and upper bound of similar degree in Phase-I, and then using structural degree and similar degree in Phase-II. The pseudocode of our two-phase vertex reduction is shown in Algorithm 11, denoted indexedVR. In Phase-I, we first obtain the structural degree $d(u)$ for each $u \in V_L \cup V_R$ (Line 1), and an upper bound $\bar{\delta}(u)$ of the similar degree for each vertex $u \in V_L$ (Line 2). Then, as long as there is a non-deleted vertex $u \in V_L \cup V_R$ satisfying $d(u) < \tau$ or a non-deleted vertex $u \in V_L$ satisfying $\bar{\delta}(u) < \tau - 1$ (Line 3), we mark u as deleted and update the structural degree of its structural neighbors (Lines 4–5); note that, we do not update $\bar{\delta}(\cdot)$ in Phase-I. In Phase-II, we first compute a *progressive* similar degree, denoted $\delta_p(\cdot)$, for each non-deleted L-side vertex, by invoking progressiveSN (Lines 7–8). Here, $\delta_p(u)$ is a lower bound of u's similar degree $\delta(u)$, and it records the number of similar neighbors that have been computed for u; our computation of $\delta_p(u)$ ensures that $\delta_p(u) \geq \tau - 1$ if and only if $\delta(u) \geq \tau - 1$. Then, as

long as there is a non-deleted vertex $u \in V_L \cup V_R$ satisfying $d(u) < \tau$ or a non-deleted vertex $u \in V_L$ satisfying $\delta_p(u) < \tau - 1$, we mark u as deleted (Line 18) and update the structural degree of its structural neighbors (Line 10). Furthermore, if u is an L-side vertex, we also obtain the set $\Gamma(u)$ of similar neighbors of u (Line 12), and update the progressive similar degree $\delta_p(v)$ to satisfy the invariant that $\delta_p(v) \geq \tau - 1$ if and only if $\delta(v) \geq \tau - 1$ for each $v \in \Gamma(u)$ (Lines 13–17). Note that, in our implementation, we use a queue to maintain the vertices that satisfy the condition at Line 3 or Line 9; as a result, we do not need to loop through all non-deleted vertices to find the unpromising vertices.

In Algorithm 11, for an L-side vertex u, we compute $\delta_p(u)$ instead of $\delta(u)$. Our main motivation is that for an L-side vertex u satisfying $d(u) \geq \tau$, we only need to compute $\tau - 1$ of its similar neighbors to certify that it is a promising vertex. That is, we stop the computation of $\Gamma(u)$ once $\delta_p(u) \geq \tau - 1$; however, if some of the computed similar neighbors of u are later removed (i.e., marked as deleted), then we need to update $\delta_p(u)$ by computing more similar neighbors of u (Lines 15–17 of Algorithm 11). As a result, for vertices with high similar degrees in the remaining graph (i.e., obtained by removing all unpromising vertices), we only need to compute a small portion of their similar neighbors to prevent them from being removed and thus save unnecessary similar neighbor computations. The pseudocode of computing $\delta_p(u)$ is shown in Lines 19–26 of Algorithm 11, denoted progressiveSN. It is invoked only when $\delta_p(u) < \tau - 1$ and there are still unchecked segments of Φ_u. In progressiveSN, we check the segments of $\mathbb{S}_u$ one by one (Line 20–24), and stop once we have found enough similar neighbors for u (Line 25). We record the index of the first unchecked segment in $\mathsf{idx}(u)$ (Line 8).

indexedVR is better than VReduce (Algorithm 8), since (1) Phase-I of indexedVR is lightweight but very effective at pruning vertices as demonstrated by our empirical studies, and (2) indexedVR uses indexedSN and progressiveSN to compute the similar neighbors.

Overall Algorithm. Our index-based MSBE improves upon Algorithm 7 by replacing the invocation to VReduce at Line 2 with invoking indexedVR for vertex reduction, and invokes indexedSN to compute $\Gamma(u)$ at Lines 6 and 15. Nevertheless, the time complexity of index-based MSBE remains $\mathcal{O}(|V_L| \cdot |E| \cdot 2^{|V_L|})$ as proved in Theorem 4, by noting that the time complexity of indexedSN remains $\mathcal{O}(|E|)$. Despite of having the same time complexity, our empirical studies in Section 4.7 show that the index-based approach can improve the efficiency of MSBE by several orders of magnitude.

4.4.3 Index Construction

In this subsection, we present two algorithms to construct the index based on the ideas of *largest gap* and *steady segment*, respectively. Note that, the indexes are constructed offline, and once constructed, they can be used to process maximal similar-biclique enumeration queries with different ε and τ values.

Largest Gap (LG) Index. Recall that, our index structure summarizes a subset of vertices of Φ_u and their similarities to a vertex u by four numbers $\mathsf{seg} = \langle \mathsf{v}_{\min}, \mathsf{v}_{\max}, \mathsf{s}_{\max}, c \rangle$, where $\mathsf{s}_{\max}$ is an upper bound of the similarity between u and

each $v \in \Phi_u$ such that $\mathsf{v}_{\min} \leq v \leq \mathsf{v}_{\max}$. To obtain the similar neighbors of u that are in the range $[\mathsf{v}_{\min}, \mathsf{v}_{\max}]$, we need to go through each vertex $v \in [\mathsf{v}_{\min}, \mathsf{v}_{\max}]$ and test its similarity with u (e.g., see Line 3 of Algorithm 10) even if $v \notin V(\mathsf{seg})$. We call a vertex v that is in the range $[\mathsf{v}_{\min}, \mathsf{v}_{\max}]$ but not in $V(\mathsf{seg})$ a *fake vertex*.

Intuitively, we should minimize the number of fake vertices when constructing the index. We call the index built by this strategy the largest gap (LG) index. It is constructed as follows. Suppose we are going to cover Φ_u by k segments. This is equivalent to find $k-1$ cut points in the sequence of vertices of Φ_u that are ordered in increasing vertex id order; denoted the sequence of vertices as $\{v_1, v_2, \ldots, v_{|\Phi_u|}\}$. Note that, the vertex ids are not consecutive, i.e., it is possible that $v_2 - v_1 > 1$. We represent the $k-1$ cut points by $k-1$ index values $\{\ell_1, \cdots, \ell_{k-1}\}$ such that $1 < \ell_1 < \ell_2 < \cdots < \ell_{k-1} < |\Phi_u|$, i.e., the i-th cut point is between vertex $v_{\ell_i - 1}$ and vertex v_{ℓ_i}. Define $\ell_0 = 1$ and $\ell_k = |\Phi_u| + 1$. Then, segment seg_i covers vertices $\{v_{\ell_{i-1}}, \ldots, v_{\ell_i - 1}\}$, for $1 \leq i \leq k$. Let f_1 be the number of fake vertices if we cover Φ_u by only one segment, i.e., $f_1 = v_{|\Phi_u|} - v_1 + 1 - |\Phi_u|$ where $v_{|\Phi_u|} - v_1 + 1$ is the total number of vertices in the range $[v_1, v_{|\Phi_u|}]$. It is easy to verify that the number of fake vertices of covering Φ_u by k segments with the $k-1$ cut points $\{\ell_1, \ldots, \ell_{k-1}\}$ is $f_1 - \sum_{i=1}^{k-1}(v_{\ell_i} - v_{\ell_i - 1} + 1)$. As f_1 is a fixed number for Φ_u, minimizing the number of fake vertices is equivalent to maximizing $\sum_{i=1}^{k-1}(v_{\ell_i} - v_{\ell_i - 1} + 1)$.

Definition 7 (Gap)**.** Given the sequence of vertices $\{v_1, v_2, \ldots, v_{|\Phi_u|}\}$ of Φ_u that are ordered in increasing vertex id order, the *gap* of vertex v_i for $i > 1$ is defined as $v_i - v_{i-1} + 1$; the gap of v_1 is defined as 0.

Thus, the LG index constructs k segments to cover Φ_u, where the $k-1$ cut points are the $k-1$ vertices with the largest gaps. We omit the details, since it is outperformed by our steady segment index as introduced next.

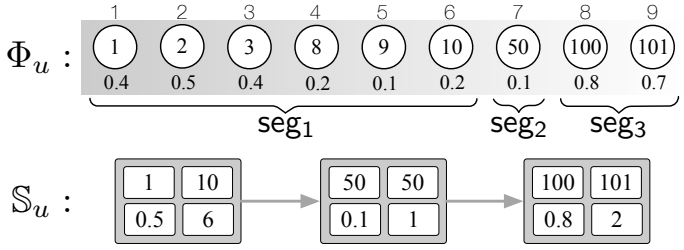


Figure 4.5: Example of consLG

Example 11. *Figure 4.5 shows the three segments constructed by the largest gap strategy for the same Φ_u as Example 10.*

Choosing the number of segments to cover Φ_u. It is easy to see that the more the number of segments, the fewer the number of fake vertices introduced by the segments. In the extreme case of covering Φ_u by $|\Phi_u|$ segments, there will be no fake vertices introduced. However, the space complexity would be too high to be practical, as discussed in Section 4.4.1. Thus, we set the number of segments for covering Φ_u as $\alpha \cdot \log |\Phi_u|$ where α is a user defined parameter, in viewing that a fixed number for different Φ_u will not work as $|\Phi_u|$ varies a lot across different vertices u.

Steady Segment (SS) Index. The LG index ignores the similarities (between u and different vertices) in a segment, and thus may result in a very wide range of similarity values for a segment. This is not good for indexedSN and progressiveSN, as they need to check all vertices covered by a segment seg even if there is only one vertex in seg whose similarity to u is no lower than ε. Motivated by this, we aim to construct *steady segments* such that all similarities in a segment are close to each other.

Definition 8 (Steady Segment)**.** Given a steady threshold $0 < \gamma < 1$, a segment $\mathsf{seg} = \langle \mathsf{v}_{\min}, \mathsf{v}_{\max}, \mathsf{s}_{\max}, c \rangle$ of Φ_u is steady if $\max_{v \in V(\mathsf{seg})} \mathsf{sim}(u, v) - \min_{v \in V(\mathsf{seg})} \mathsf{sim}(u, v) \leq \gamma$.

The first term, $\max_{v \in V(\mathsf{seg})} \mathsf{sim}(u, v)$, is exactly $\mathsf{seg}.\mathsf{s}_{\max}$. For ease of presentation, we denote the second term, $\min_{v \in V(\mathsf{seg})} \mathsf{sim}(u, v)$, by $\mathsf{seg}.\mathsf{s}_{\min}$, the smallest similarity value. A segment seg is steady if $\mathsf{seg}.\mathsf{s}_{\max} - \mathsf{seg}.\mathsf{s}_{\min} \leq \gamma$. The main advantage of a steady segment is that if seg is steady and satisfies $\mathsf{seg}.\mathsf{s}_{\max} \geq \varepsilon$, then it is likely that many vertices of $V(\mathsf{seg})$ have similarity values to u no lower than ε, and thus most of the computation will not be wasted.

Ideally, we would like to find the minimum number of steady segments to cover Φ_u. However, the number of required steady segments could be very large. For example, if the steady threshold γ is very close to 0 and all vertices of Φ_u have different similarity values to u, then the number of required steady segments to cover Φ_u is $|\Phi_u|$. Thus, we instead construct a fixed number of steady segments to cover as many vertices of Φ_u as possible, and then cover the remaining uncovered vertices of Φ_u by as few segments as possible by ignoring the difference between the similarity values.

Given γ and k, our problem is to find k steady segments to cover as many vertices of Φ_u as possible. We first construct, for each vertex $v \in \Phi_u$, a maximal steady segment seg_v that starts at v (i.e. $\mathsf{seg}_v.\mathsf{v}_{\min} = v$), and then select k segments $\mathbb{S}^*$ from $\{\mathsf{seg}_v \mid v \in \Phi_u\}$ such that $\left|\bigcup_{v \in \mathbb{S}^*} V(\mathsf{seg}_v)\right|$ is maximized. This is an instance of the maximum k-coverage problem which is NP-hard (Megiddo et al., 1983). We select the k segments in a greedy manner. That is, the k segments are selected one-by-one. Let S be the starting vertices of the currently selected segments. Then, the next segment to be added to S is $\arg\max_{v \in \Phi_u} \left|\bigcup_{v' \in S \cup \{v\}} V(\mathsf{seg}_{v'})\right|$. As this function is submodular, the greedy approach achieves an approximation ratio of $1 - \frac{1}{e}$ (Hochbaum, 1996).

The pseudocode is shown in Algorithm 12, denoted consSS. For each vertex u, we first compute its 2-hop structural neighbors Φ_u and their similarities to u (Line 2), and sort Φ_u in increasing vertex id order (Line 3). Then, for each $v_i \in \Phi$, we compute the maximal steady segment seg_{v_i} that starts at v_i, by iteratively trying to add the next vertex to the segment (Lines 7–12). Next, we iteratively add to $\mathbb{S}_u$ the segment of $\mathbb{C}$ that covers the largest number of uncovered vertices of Φ_u (Lines 14–22). Note that, after adding a segment into $\mathbb{S}_u$, we also need to update the remaining segments of $\mathbb{C}$ to be disjoint from the segments of $\mathbb{S}_u$ (Lines 18–22). During this process, for time efficiency consideration, we do not maintain $\mathsf{seg}.\mathsf{s}_{\max}$; instead, we compute $\mathsf{seg}.\mathsf{s}_{\max}$ for each segment $\mathsf{seg} \in \mathbb{S}_u$ later (Line 23). Finally,

Algorithm 12: consSS$(G = (V_L, V_R, E), \alpha, \gamma)$

```
1  for each u ∈ V_L ∪ V_R do
2      Φ_u ← SimNei(G, u, 1/(2|V_L|+2|V_R|));
3      Let {v_1, v_2, ..., v_{|Φ_u|}} be vertices of Φ_u in increasing vertex id order;
4      ℂ ← ∅;
5      for i ← 1 to |Φ_u| do
6          s_min ← 1;   s_max ← 0;
7          for j ← i to |Φ_u| do
8              if sim(u, v_j) < s_min then s_min ← sim(u, v_j);
9              if sim(u, v_j) > s_max then s_max ← sim(u, v_j);
10             if s_max − s_min > γ then
11                 seg_{v_i} ← ⟨v_i, v_{j−1}, null, j − i⟩;
12                 break;
13         ℂ ← ℂ ∪ {seg_{v_i}};
14     𝕊_u ← ∅;   k ← min{|Φ_u|, α · log |Φ_u|};
15     while |𝕊_u| < k and ℂ ≠ ∅ do
16         seg* ← arg max_{seg∈ℂ} seg.c;
17         𝕊_u ← 𝕊_u ∪ {seg*};
18         for each seg ∈ ℂ do
19             if seg*.v_min < seg.v_min ≤ seg*.v_max then
20                 Remove seg from ℂ
21             else if seg.v_min < seg*.v_min ≤ seg.v_max then
22                 Let v be the vertex that immediately precedes seg*.v_min in Φ_u, change
                    seg.v_max to be v, and update seg.c accordingly in ℂ;
23     for each seg ∈ 𝕊_u do Compute seg.s_max;
24     for each maximal consecutive sequence of vertices v_i, v_{i+1}, ..., v_j of Φ_u that are not
         covered by 𝕊_u do
25         Add to 𝕊_u the segment that covers {v_i, ..., v_j};
26 return ℐ = {𝕊_u | u ∈ V_L ∪ V_R};
```

we create the minimum number of segments to cover all vertices of Φ_u that are not covered by $\mathbb{S}_u$ (Lines 24–25).

Example 12. *Figure 4.6 shows the three steady segments constructed for the same Φ_u in Examples 10 and 11, where $\gamma = 0.1$.*

Similarity Tree. Lines 5–12 of Algorithm 12, which constructs the initial maximal steady segments for each vertex, has a high time complexity of $O(|\Phi_u|^2)$, and may dominate the total running time of Algorithm 12. In view of this, we build a similarity tree data structure $\mathcal{T}_u$ for each Φ_u to speed up the process. $\mathcal{T}_u$ is similar to a range tree or segment tree (de Berg et al., 2008). Each tree node t of $\mathcal{T}_u$ represents a range of vertices of Φ_u — specifically, the vertices corresponding to the leaf nodes of the subtree rooted at t — and records two values $t.\mathsf{s}_{\min}$ and $t.\mathsf{s}_{\max}$ which are, respectively, the smallest similarity and the largest similarity among the vertices represented by t. An example similarity tree is shown in Figure 4.7. t_5 represents the third vertex and the forth vertex, while t_3 represents the last four vertices of Φ_u. Let $\{v_1, v_2, \ldots, v_{|\Phi_u|}\}$ be the vertices of Φ_u sorted in increasing id order. We first create one leaf node t for each vertex $v \in \Phi_u$ with $t.\mathsf{s}_{\min} = t.\mathsf{s}_{\max} = \mathsf{sim}(u, v)$. Then,

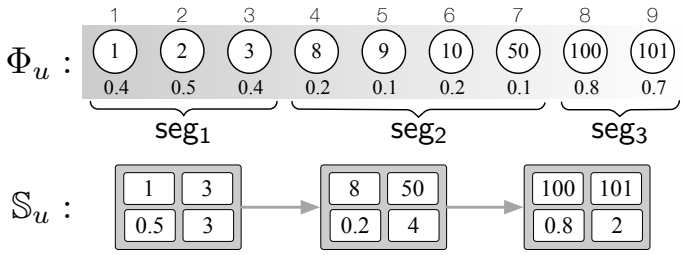


Figure 4.6: Example of consSS

we construct the tree layer-by-layer in a bottom-up manner. Let T_i be the list of tree nodes at the current layer. We go through T_i by accessing two tree nodes each time. For each pair of tree nodes t and t', we create a new tree node t_p as their parent and put t_p into list T_{i+1}; if there is only one node in the last step, we directly put it into T_{i+1}. Note that, $t_p.\mathsf{s}_{\min} = \min\{t.\mathsf{s}_{\min}, t'.\mathsf{s}_{\min}\}$ and similarly $t_p.\mathsf{s}_{\max}$. The construction finishes when a layer has only one node, which is the root of the similarity tree. It is easy to see that tree construction takes $O(|\Phi_u|)$ time, as the tree is a complete binary tree.

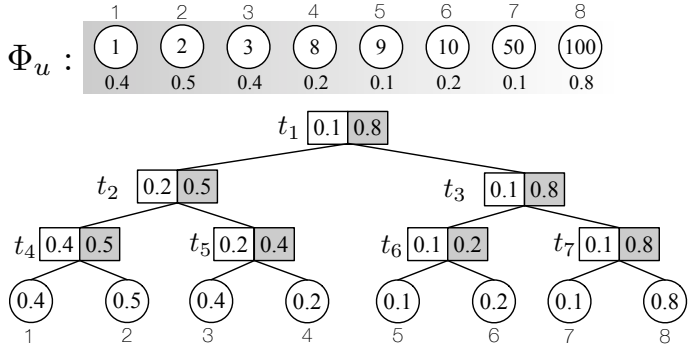


Figure 4.7: Similarity tree data structure

To compute the maximal steady segment of v, we first traverse the similarity tree upwards, starting from the leaf node that corresponds to v, and then go downwards. During the process, we maintain $\mathsf{s}_{\min}$ and $\mathsf{s}_{\max}$, which are initialized by $\mathsf{sim}(u, v)$. In the upward phase, if the current tree node t is a right child of its parent, then we directly go to its parent. Otherwise, let t' be the right-sibling of t. If $\max\{\mathsf{s}_{\max}, t'.\mathsf{s}_{\max}\} - \min\{\mathsf{s}_{\min}, t'.\mathsf{s}_{\min}\} \leq \gamma$ which means that we can include all vertices represented by t' into the segment, then we update $\mathsf{s}_{\min}$ and $\mathsf{s}_{\max}$ by $\min\{\mathsf{s}_{\min}, t'.\mathsf{s}_{\min}\}$ and $\max\{\mathsf{s}_{\max}, t'.\mathsf{s}_{\max}\}$, respectively, and go to its parent. Otherwise, we go to t' and move into the downward phase. In the downward phase, let t_l be the left child of the current node t. If $\max\{\mathsf{s}_{\max}, t_l.\mathsf{s}_{\max}\} - \min\{\mathsf{s}_{\min}, t_l.\mathsf{s}_{\min}\} \leq \gamma$, then we update $\mathsf{s}_{\min}$ and $\mathsf{s}_{\max}$ and go to t's right child. Otherwise, we go to t's left child. Finally, when we arrive at a leaf node, we can decide whether the corresponding vertex should be included into the segment or not. It is easy to see that this process takes $O(\log |\Phi_u|)$ time which is the height of the similarity tree. Thus, constructing the maximal steady segment for all vertices of Φ_u takes $O(|\Phi_u| \log |\Phi_u|)$ time.

Example 13. *Suppose we are going to construct the maximal steady segment for the second vertex of Φ_u as shown in Figure 4.7 with $\gamma = 0.3$. Initially, $\mathsf{s}_{\min} = \mathsf{s}_{\max} = 0.5$ and t is the second leaf node. As t is a right child of its parent t_4, we directly go to t_4. Now, t_4 is a left child of its parent t_2, and t_4's right sibling is t_5. As $\max\{\mathsf{s}_{\max}, t_5.\mathsf{s}_{\max}\} - \min\{\mathsf{s}_{\min}, t_5.\mathsf{s}_{\min}\} = 0.5 - 0.2 = 0.3 \leq \gamma$, we include all vertices represented by t_5 (i.e., the third and forth vertices) into the segment, update $\mathsf{s}_{\min}$ to be $\max\{\mathsf{s}_{\min}, t_5.\mathsf{s}_{\min}\} = 0.2$ and $\mathsf{s}_{\max}$ to be 0.5, and then go to its parent t_2. t_2 is a left child of its parent and its right sibling is t_3. As $\max\{\mathsf{s}_{\max}, t_3.\mathsf{s}_{\max}\} - \min\{\mathsf{s}_{\min}, t_3.\mathsf{s}_{\min}\} = 0.8 - 0.1 > \gamma$, we go to t_3 and move into the downward phase. t_3's left child is t_6 and $\max\{\mathsf{s}_{\max}, t_6.\mathsf{s}_{\max}\} - \min\{\mathsf{s}_{\min}, t_6.\mathsf{s}_{\min}\} = 0.5 - 0.1 > \gamma$, we go to its left child t_6. Similarly, we go to t_6's left child, which is a leaf node corresponding to the fifth vertex of Φ_u. We find that the first vertex cannot be included into the segment. Thus, the maximal steady segment consists of three vertices, the second, third, and forth vertices.*

Computing maximal steady segments in linear time. Although similarity tree can facilitate the computation of maximal steady segment, we observe in experiments that this process is still very time-consuming. To further improve the efficiency, we propose a two-pointers based method to construct the initial maximal steady segments for each vertex in Φ_u, whose time complexity is linear to the size of Φ_u, i.e., $O(|\Phi_u|)$. In this subsection, we denote the maximal steady segment of a vertex u by $\tilde{\mathsf{seg}}_u$. Firstly, we have the following observation.

Observation 1. Suppose Φ_u is sorted in increasing order of vertex id, i.e., $\{v_1, v_2, \ldots, v_{|\Phi_u|}\}$. For two vertices $v_i, v_j \in \Phi_u$ s.t. $i < j$, the largest vertex id in the maximal steady segment of v_j is larger than that of v_i, i.e., $\tilde{\mathsf{seg}}_{v_j}.\mathsf{v}_{\max} > \tilde{\mathsf{seg}}_{v_i}.\mathsf{v}_{\max}$.

Suppose vertices in Φ_u are sorted in increasing order of vertex id and we process each vertex one by one. Starting from v_1, we create two pointers i and j that both point to v_1 initially. Then, we move j to the next vertex along this order until the segment formed by i and j is not steady or j reaches the last vertex in Φ_u. Here, the segment formed by i and j means this segment covers vertices in the range from i to j in Φ_u. To check if the segment formed by i and j is steady or not, we maintain two queues Q_1 and Q_2 which record the smallest similarity and the largest similarity between u and the vertices in the range from i to j. The maintenance of Q_1 and Q_2 will be introduced later. When j reaches at a vertex that makes the segment formed by i and j not steady or j reaches the last vertex in Φ_u, we stop moving j and construct the maximal steady segment for v_1 according to i and j. Then, we start to process v_2 by making i point to v_2. According to Observation 1, we can safely continue to move j if it has not reached the last vertex. Note that during the movement of i and j, it is easy to check if the segment formed by i and j is steady or not by maintaining Q_1 and Q_2. When j reaches at a vertex that makes the segment formed by i and j not steady once again or j reaches the last vertex in Φ_u, we stop moving j and construct the maximal steady segment for v_2 according to i and j. We construct the maximal steady segments for remaining vertices in a similar way. According to the definition of maximal steady segment, it is easy to see that the above procedure can construct maximal steady segments for each vertex in Φ_u correctly.

Generally speaking, Q_1 and Q_2 are double-ended queues whose element is similarity. Elements in Q_1 (resp. Q_2) obey a non-increasing (resp. non-decreasing) order and the head element is the smallest (resp. largest) similarity between u and a vertex in the range from i to j. Initially, both i and j point to v_1 and we insert the similarity between u and v_1 into Q_1 and Q_2. When j moves to the next vertex, we do the following operations:

- For Q_1, remove the element from the end of Q_1 until the last queue element is smaller than the similarity between u and the vertex pointed by j, or Q_1 becomes empty.
- For Q_2, remove the element from the end of Q_2 until the last queue element is lager than the similarity between u and the vertex pointed by j, or Q_2 becomes empty.
- Add the similarity between u and the vertex pointed by j to the end of Q_1 and Q_2.

When i moves to the next vertex, we remove the first element in Q_1 (as well as Q_2) if its corresponding vertex in Φ_u precedes the vertex pointed by i.

In this way, we always guarantee that the first element in Q_1 (resp. Q_2) is the maximum (resp. minimum) similarity between u and a vertex in the range from i to j. It is easy to check if the current segment formed by i and j is steady or not by comparing the gap between the first element from Q_1 and Q_2 with γ.

The pseudo code is shown in Algorithm 13. The input of TPA consists of u's 2-hop neighbors Φ_u and a steady threshold γ. Note that vertices in Φ_u are in the increasing order of vertex id. Firstly, we create two pointers i and j, which are both initialized as 1. We also create two empty queues Q_1 and Q_2, in which each element is a pair of variables, in the form of $\langle s, p\rangle$. Here, s records the similarity and p records the position of the corresponding vertex in Φ_u. Then, we start to construct maximal steady segment for each vertex one by one (Lines 4-13). Specifically, i points to the vertex that is processed in the current iteration and j keeps moving to the next vertex until it reaches the end vertex of Φ_u or the segment formed by i and j is not steady (Line 5). Each time j moves to the next vertex (Line 8), we invoke procedure update to modify Q_1 (Line 6) and Q_2 (Line 7). Procedure update will be introduced later. Once we find that the segment formed by i and j is no longer steady by checking the head elements of Q_1 and Q_2 (Line 5), we terminate the movement of j and construct the maximal steady segment of v_i (Lines 9-10). Then we increase i by 1 to process next vertex (Line 11). Then we update the head element of Q_1 and Q_2 to make sure that they are both in the range from i to j. That is, we remove the head element from the queue if its position is smaller than i (Lines 12-13).

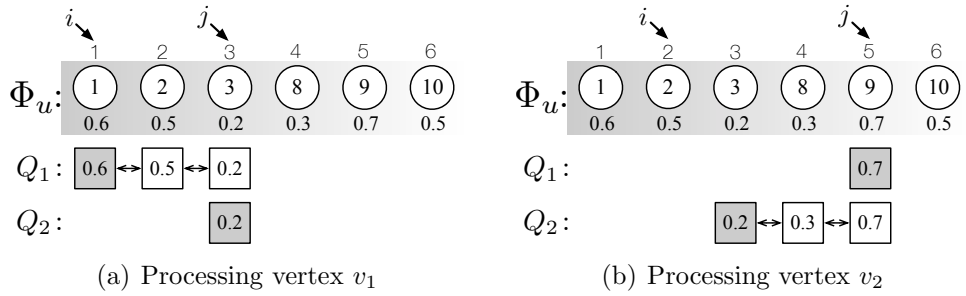
The input of procedure update consists of the queue Q, the similarity (i.e., s) and its position in Φ_u (i.e., j), and a boolean variable flag to indicate how to maintain Q. If flag is 0, we maintain elements in Q in a decreasing order with respect to the similarity (Lines 15-18). Specifically, we remove elements from the tail of the queue until the similarity of the last element is larger than s, or the queue becomes empty (Lines 16-17). Then, we add the new element $\langle s, j\rangle$ to the tail of the queue (Line

18). If flag is 1, we maintain elements in Q in an increasing order with respect to the similarity in a similar way (Lines 19-22).

Algorithm 13: TPA(Φ_u, γ)

```
1  i ← 1, j ← 1;
2  Q1 ← ∅, Q2 ← ∅;
3  ℂ ← ∅;
4  while i < |Φu| do
5      while j < |Φu| and Q1.head.first − Q2.head.first ≤ γ do
6          update(Q1, sim(u, vj), j, 0);          /* decreasing order */;
7          update(Q2, sim(u, vj), j, 1);          /* increasing order */;
8          j ← j + 1;
9      seg_vi ← ⟨vi, vj−1, Q1.head.first, j − i⟩;
10     ℂ ← ℂ ∪ {seg_vi};
11     i ← i + 1;
12     if Q1.head.second < i then Remove the head element of Q1;
13     if Q2.head.second < i then Remove the head element of Q2;
14 return ℂ;
   Procedure update(Q, s, j, flag)
15 if flag = 0 then
16     while Q.tail.first < s do
17         Remove the tail element of Q;
18     Add ⟨s, j⟩ to the tail of Q;
19 else
20     while Q.tail.first > s do
21         Remove the tail element of Q;
22     Add ⟨s, j⟩ to the tail of Q;
```

To apply TPA, we replace Lines 4-13 of Algorithm 12 by Algorithm 13. It is easy to see that the time complexity of Algorithm 13 is linear to the size of Φ_u. This is because each vertex in Φ_u is added into Q_1 (resp. Q_2) at most once and removed from Q_1 (resp. Q_2) at most once. Thus, the time complexity of Algorithm 13 is $O(|\Phi_u|)$, which is better than similarity tree based algorithm whose time complexity is $O(|\Phi_u| \log |\Phi_u|)$.



(a) Processing vertex v_1 (b) Processing vertex v_2

Figure 4.8: Two pointers algorithm

Example 14. *Suppose we are going to construct the maximal steady segments for each vertex in Φ_u as shown in Figure 4.8 with $\gamma = 0.3$. Firstly, we process v_1. We*

make i and j both point to v_1 initially, and then we move j. When j reaches at v_3 (see Figure 4.8(a)), we find that the segment formed by i and j is not steady. This can be verified by checking the head elements of Q_1 and Q_2, which are colored as grey, i.e., $0.6 - 0.2 > 0.3$. Thus, we construct the maximal steady segment for v_1 according to i and j, i.e., $\langle v_1, v_2, 0.6, 2\rangle$. Next, we start to process v_2 by making i point to v_2. At this stage, the head element in Q_1 will be removed since its corresponding vertex in Φ_u precedes the vertex pointed by i. Now the segment formed by i and j (recall that j is pointing to v_3) becomes steady since the head element in Q_1 now is 0.5 and $0.5 - 0.2 \leq 0.3$. We continue to move j. When j reaches at v_9 (see Figure 4.8(b)), we find that the segment formed by i and j is not steady once again, i.e., $0.7 - 0.2 > 0.3$. Thus, we construct the maximal steady segment for v_2 according to i and j, i.e., $\langle v_2, v_8, 0.5, 3\rangle$. We construct maximal steady segments for remaining vertices in a similar way.

Analysis of consSS. For each vertex $u \in V_L \cup V_R$, Line 2 of Algorithm 12 takes $\mathcal{O}(\sum_{v \in N(u)} d(v))$ time, Line 3 as well as Lines 5–12 take $\mathcal{O}(|\Phi_u| \log |\Phi_u|)$ time; Lines 5–12 use the similarity tree data structure as discussed above. The while loop at Line 15 runs for at most $\alpha \cdot \log |\Phi_u|$ iterations, and each iteration takes $\mathcal{O}(|\Phi_u|)$ time. Lines 23–25 take $\mathcal{O}(|\Phi_u|)$ time. Thus, the total time complexity of consSS is $\mathcal{O}\Big(\sum_{u \in V_L \cup V_R} \big(\alpha |\Phi_u| \log |\Phi_u| + \sum_{v \in N(u)} d(v)\big)\Big)$ time.

4.5 Index Maintenance

In the previous section, we introduced a novel index and index construction algorithms. However, most of real-world graphs are frequently updated. Motivated by this issue, in this section we discuss algorithms for maintaining our proposed index structure when graphs update. Note that the update algorithms introduced in this section can both be applied to LG index and SS index.

Here, we mainly focus on the edge insertion and deletion, as the vertex updates can be handled by performing several edge updates. Note that our index is relevant to the vertex id. When deleting a vertex, we preserve its id and do not assign it to the later new vertices. When inserting a new vertex, its id is set as the current largest id plus 1, (i.e., $n + 1$, where n is the current largest id in the graph).

4.5.1 Edge Insertion

Let (u, v) be an edge inserted into G. W.l.o.g., assume u is an R-side vertex and v is an L-side vertex. Then, we have the following observation.

Observation 2. Suppose a new edge (u, v) is inserted, then only the index structure (i.e., $\mathbb{S}_w$) for $w \in \Phi_u \cup \Phi_v \cup \{u, v\}$ will change.

Firstly, let us discuss the index maintenance for R-side vertices, i.e., Φ_u and u. For vertices $w \in \Phi_u$, $\mathbb{S}_w$ needs to be updated since the similarity between u and w may change. In this case, we find out the segment in $\mathbb{S}_w$ that covers u (i.e., $\mathsf{seg}_i \in \mathbb{S}_w$ s.t. $\mathsf{seg}_i.\mathsf{v}_{\min} \leq u \leq \mathsf{seg}_i.\mathsf{v}_{\max}$) and update $\mathsf{seg}_i.\mathsf{s}_{\max}$ as $\max\{\mathsf{sim}(u, w), \mathsf{seg}_i.\mathsf{s}_{\max}\}$. At the same time, we need to increase $\mathsf{seg}_i.c$ by 1 if u is a new 2-hop neighbor of

w, which can be verified by checking whether $\mathsf{ComNei}(u, w) = 1$ and $(v, w) \in E$ or not. However, if we cannot find the segment in $\mathbb{S}_w$ that covers u, this means u must be a new 2-hop neighbor of w. We create a new segment of u and insert this new segment into $\mathbb{S}_w$. The above update process is correct, since we always guarantee that segments in $\mathbb{S}_w$ cover all w's 2-hop neighbors and each segment has correct $\mathsf{s}_{\max}$ and c. For vertex u, $\mathbb{S}_u$ can be updated in a similar way. For each vertex $w \in \Phi_u$, we find out the segment in $\mathbb{S}_u$ that covers w (i.e., $\mathsf{seg}_i \in \mathbb{S}_u$ s.t. $\mathsf{seg}_i.\mathsf{v}_{\min} \leq w \leq \mathsf{seg}_i.\mathsf{v}_{\max}$) and update $\mathsf{seg}_i.\mathsf{s}_{\max}$ as $\max\{\mathsf{sim}(u, w), \mathsf{seg}_i.\mathsf{s}_{\max}\}$. At the same time, we need to increase $\mathsf{seg}_i.c$ by 1 if w is a new 2-hop neighbor of u. However, if we cannot find the segment in $\mathbb{S}_u$ that covers w, we create a new segment of w and insert this new segment into $\mathbb{S}_u$.

Optimization. In the above process, for each $w \in \Phi_u$ we need to compute $\mathsf{sim}(u, w)$. If $\mathsf{sim}(u, w)$ is larger than their original similarity before the edge insertion, then the $\mathsf{s}_{\max}$ of the segment in $\mathbb{S}_u$ that covers w and of the segment in $\mathbb{S}_w$ that covers u may be updated. However, if we know in advance that $\mathsf{sim}(u, w)$ will not increase after edge insertion, we do not need to compute $\mathsf{sim}(u, w)$ and update the segments. The following lemma describes this optimization.

Lemma 11. *Suppose w and u are 2-hop neighbors with each other in the original graph. After inserting (u, v), the similarity between w and u increases if $(w, v) \in E$, decreases otherwise.*

Proof. Assume the degrees of u and w in the original graph are $d(u)$ and $d(w)$, respectively. The number of common neighbors between u and w is C in the original graph. After inserting (u, v), the degree of u will increase by 1. If $(w, v) \in E$, the number of common neighbors between u and w will increase by 1. Thus, the similarity between u and w is $\mathsf{sim}(u, w) = \frac{C+1}{d(u)+d(w)-C}$, which is larger than their original similarity, i.e., $\frac{C}{d(u)+d(w)-C}$. If $(w, v) \notin E$, the number of common neighbors between u and w will be same and $\mathsf{sim}(u, w) = \frac{C}{d(u)+d(w)-C+1}$, which is smaller than their original similarity. □

According to Lemma 11, before we compute the similarity between u and $w \in \Phi_u$, we can firstly check if $(w, v) \in E$ in constant time. If $(w, v) \notin E$, we know the similarity between u and w decreases and thus do not need to compute their exact similarity.

By now, we have discussed the index maintenance of edge insertion for R-side vertices. Since the two sides of a bipartite graph are interchangeable, the index maintenance for L-side vertices, i.e., $w \in \Phi_v$ and v, is same. We omit the details.

The pseudo code of edge insertion update is shown in Algorithm 14, which takes input the bipartite graph G, two end nodes u, v of the inserted edge and the index $\mathcal{I}$. W.l.o.g., we assume $u \in V_R$ and $v \in V_L$. Firstly, we compute u's 2-hop neighbors Φ_u (Line 2). Then, for each vertex $w \in \Phi_u$, if $(w, v) \notin E$ we know that $\mathsf{sim}(w, u)$ must decrease (according to Lemma 11) and we do nothing (Lines 4-5); otherwise we update the index $\mathbb{S}_w$ (Lines 6-11) and index $\mathbb{S}_u$ (Lines 12-17). Specifically, we search for the segment in $\mathbb{S}_w$ that covers u and update $\mathsf{seg}_i.\mathsf{s}_{\max}$ if necessary (Lines 6-7). At the same time, we can know if u is w's new 2-hop neighbor by checking the number of common neighbors between u and w. Here, the number of common

neighbors between u and w can be computed in $\mathsf{SimNei}(\cdot)$ (i.e., Line 2). Thus, if the number of common neighbors between u and w is 1, we increase $\mathsf{seg}_i.c$ by 1 (Line 8). Note that even if we can find out a segment covering u, u may still be w's new 2-hop neighbor because segment contains fake vertices. If we cannot find a segment covering u, this means u is w's new 2-hop neighbor. We create a segment for u and add this segment to $\mathbb{S}_w$ directly (Lines 10-11). Next, we search for the segment in $\mathbb{S}_u$ that covers w and update its $\mathsf{s}_{\max}$ and c in a similar way (Lines 13-14). If there is no segment covering w, we create a new segment for w and add it into $\mathbb{S}_u$ (Lines 16-17). The index maintenance for L-side side vertices (i.e., $\Phi_v \cup \{v\}$) is same, we just apply Lines 2-17 of Algorithm 14 by switching u and v.

Algorithm 14: EdgeInsert$^*(G = (V_L, V_R, E), u, v, \mathcal{I})$

```
1  Inserting the new edge (u, v) into G;          /* Suppose u ∈ V_R and v ∈ V_L */;
   /* Index maintenance for R-side vertices                                    */
2  Φ_u ← SimNei(G, u, 1/(2|V_L|+2|V_R|));
3  for each w ∈ Φ_u do
4      if (w, v) ∉ E then
5          continue;                          /* sim(u, w) must decrease */;
6      if ∃seg_i ∈ 𝕊_w s.t. seg_i.v_min ≤ u ≤ seg_i.v_max then
7          seg_i.s_max ← max{sim(u, w), seg_i.s_max};
8          if |ComNei(u, w)| = 1 then seg_i.c ← seg_i.c + 1;
9      else
10         seg ← ⟨u, u, sim(u, w), 1⟩;
11         𝕊_w ← 𝕊_w ∪ {seg};
12     if ∃seg_i ∈ 𝕊_u s.t. seg_i.v_min ≤ w ≤ seg_i.v_max then
13         seg_i.s_max ← max{sim(u, w), seg_i.s_max};
14         if |ComNei(u, w)| = 1 then seg_i.c ← seg_i.c + 1;
15     else
16         seg ← ⟨w, w, sim(u, w), 1⟩;
17         𝕊_u ← 𝕊_u ∪ {seg};
   /* Index maintenance for L-side vertices                                    */
18 Lines 2-17 by switching u and v;
```

4.5.2 Edge Deletion

Let (u, v) be an edge that will be deleted from G. W.l.o.g., assume u is an R-side vertex and v is an L-side vertex. Then, we have the following observation.

Observation 3. Suppose an edge (u, v) is deleted, then only the index structure (i.e., $\mathbb{S}_w$) for $w \in \Phi_u \cup \Phi_v \cup \{u, v\}$ will change.

Firstly, let us discuss the index maintenance for R-side vertices, i.e., Φ_u and u. For vertices $w \in \Phi_u$, there must be a segment $\mathsf{seg}_i \in \mathbb{S}_w$ that covers u, i.e., $\mathsf{seg}_i.\mathsf{v}_{\min} \leq u \leq \mathsf{seg}_i.\mathsf{v}_{\max}$. This is because deleting edge (u, v) will not bring a new 2-hop neighbor to w. Thus, we only need to update $\mathsf{seg}_i.\mathsf{s}_{\max}$ as $\max\{\mathsf{sim}(u, w), \mathsf{seg}_i.\mathsf{s}_{\max}\}$. For vertex u, $\mathbb{S}_u$ can be updated in a similar way. That is, for each vertex $w \in \Phi_u$, we find the segment $\mathsf{seg}_i \in \mathbb{S}_u$ that covers w and update $\mathsf{seg}_i.\mathsf{s}_{\max}$ as $\max\{\mathsf{sim}(u, w), \mathsf{seg}_i.\mathsf{s}_{\max}\}$.

Note that in the index maintenance of edge deletion, we do not update c value in each segment.

In the above process, for each $w \in \Phi_u$ we need to compute $\mathsf{sim}(u, w)$. If $\mathsf{sim}(u, w)$ is larger than their original similarity after the edge deletion, then the $\mathsf{s}_{\max}$ of the segment in $\mathbb{S}_u$ that covers w and of the segment in $\mathbb{S}_w$ that covers u may be updated. However, if we know in advance that $\mathsf{sim}(u, w)$ will not increase after the edge deletion, we do not need to compute $\mathsf{sim}(u, w)$ and update the segments. The following lemma describes this optimization.

Lemma 12. *Suppose w and u are still 2-hop neighbors with each other after deleting edge (u, v). The similarity between w and u decreases if $(w, v) \in E$, increases otherwise.*

Proof. Assume the degrees of u and w in the original graph are $d(u)$ and $d(w)$, respectively. The number of common neighbors between u and w is C in the original graph. After deleting (u, v), the degree of u will decrease by 1. If $(w, v) \in E$, the number of common neighbors between u and w will decrease by 1. Thus, the similarity between u and w is $\mathsf{sim}(u, w) = \frac{C-1}{d(u)+d(w)-C}$, which is smaller than their original similarity, i.e., $\frac{C}{d(u)+d(w)-C}$. If $(w, v) \notin E$, the number of common neighbors between u and w will be same and $\mathsf{sim}(u, w) = \frac{C}{d(u)+d(w)-C-1}$, which is larger than their original similarity. □

According to Lemma 12, before we compute the similarity between u and $w \in \Phi_u$, we can firstly check if $(w, v) \in E$ in constant time. If $(w, v) \in E$, we know the similarity between u and w decreases and thus do not need to compute their exact similarity.

By now, we have discussed index maintenance of edge deletion for R-side vertices. Since the two sides of a bipartite graph are interchangeable, the index maintenance for L-side vertices, i.e., $w \in \Phi_v$ and v, is same. We omit the details.

The pseudo code is shown in Algorithm 15. Firstly, we compute u's 2-hop neighbors Φ_u (Line 2). Then, for each vertex $w \in \Phi_u$, if $(w, v) \in E$ we know that $\mathsf{sim}(w, u)$ must decrease (according to Lemma 12) and we do nothing (Lines 4-5); otherwise we update the index $\mathbb{S}_w$ and index $\mathbb{S}_u$ (Lines 6-9). Specifically, we search for the segment in $\mathbb{S}_w$ that covers u and update its $\mathsf{s}_{\max}$ (Lines 6-7). Then, we search for the segment in $\mathbb{S}_u$ that covers w and update its $\mathsf{s}_{\max}$ (Lines 8-9). The index maintenance for L-side side vertices (i.e., $\Phi_v \cup \{v\}$) is same, we just apply Lines 2-9 of Algorithm 15 by switching u and v.

Discussion. To achieve high efficiency, our index maintenance strategy does not guarantee that the updated index is exactly same as the index constructed from scratch. For example, after inserting an edge (u, v), when we cannot find a segment in $\mathbb{S}_u$ that covers u's new 2-hop neighbor w, we directly create a new segment for w and insert this segment into $\mathbb{S}_u$. After deleting an edge, we do not update c value in each segment. The effectiveness of our index may deteriorate with the increasing number of updates, e.g., the tightness of the upper bound of similar degree. Thus, we propose to thoroughly rebuild the index from scratch when enough number of updates has occurred. To achieve this goal, we create a counter for each index to record how many updates have been occurred, including vertex insertion/deletion

Algorithm 15: EdgeDelete*$(G = (V_L, V_R, E), u, v, \mathcal{I})$

```
1  Deleting the edge (u, v) from G;          /* Suppose u ∈ V_R and v ∈ V_L */;
   /* Index maintenance for R-side vertices                                 */
2  Φ_u ← SimNei(G, u, 1/(2|V_L|+2|V_R|));
3  for each w ∈ Φ_u do
4      if (w, v) ∈ E then
5          continue;                       /* sim(u, w) must decrease */;
6      Obtain the seg_i ∈ 𝕊_w s.t. seg_i.v_min ≤ u ≤ seg_i.v_max ;
7      seg_i.s_max ← max{sim(u, w), seg_i.s_max};
8      Obtain the seg_i ∈ 𝕊_u s.t. seg_i.v_min ≤ w ≤ seg_i.v_max ;
9      seg_i.s_max ← max{sim(u, w), seg_i.s_max};
   /* Index maintenance for L-side vertices                                 */
10 Lines 2-9 by switching u and v;
```

and edge insertion/deletion. We rebuild the index from scratch if this number is larger than $\zeta \times |E|$, where ζ is the parameter to control the frequency of index rebuilding and $|E|$ is the number of edges in the corresponding bipartite graph. In experiments (i.e., Section 4.7.3), we will explain how to choose an appropriate value of ζ. Note that, when we rebuild the index from scratch, we reassign the vertex id to make sure that vertices of V_L take (integer) ids from $\{1, 2, \ldots, |V_L|\}$, and vertices of V_R take ids from $\{1 + |V_L|, 2 + |V_L|, \ldots, |V_R| + |V_L|\}$.

4.6 Parallelization

In this section, we parallelize our index construction algorithm consSS**. Note that other index construction algorithms can also be parallelized in a similar way. In consSS**, the most time-consuming part is computing the set of segments, which is an independent procedure for each vertex $u \in V_L \cup V_R$. Motivated by this, we propose to speed up the index construction with shared-memory parallelization (i.e., handling different vertices simultaneously with multiple threads). Specifically, the "for loop" at Line 1 of Algorithm 12 can be parallelized. We use openMP to parallelize this "for loops" in our parallel implementation.

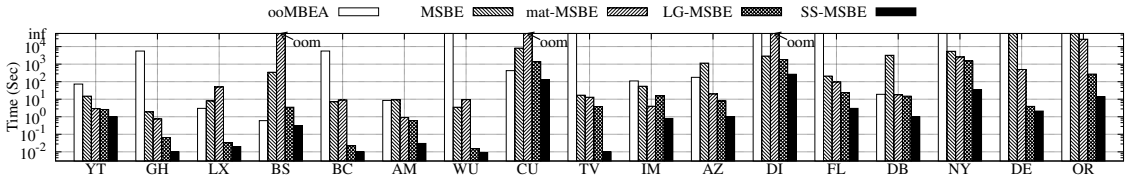


Figure 4.9: Running time on all graphs ($\varepsilon = 0.5$, $\tau = 3$)

4.7 Experiments

In this section, we evaluate the efficiency of our algorithms as well as the effectiveness of our similar-biclique model.

Algorithms. We compare the following algorithms.

Table 4.1: Statistics of graphs

Abbreviation	Graph	$\|V_L\|$	$\|V_R\|$	$\|E\|$	Type
YT	YouTube	94,238	30,087	293,360	Membership
GH	GitHub	56,519	120,867	440,237	Membership
LX	Linux	42,045	337,509	599,858	Post
BS	Bibsonomy	767,447	5,794	801,784	Assignment
BC	BookCross	105,278	340,523	1,149,739	Rating
AM	ActorMovie	127,823	383,640	1,470,404	Appearance
WU	WebUni	6,202	200,148	1,948,004	Appearance
CU	CiteULike	731,769	153,277	2,338,554	Assignment
TV	TVTropes	64,415	87,678	3,232,134	HasFeature
IM	IMDB	303,617	896,302	3,782,463	Appearance
AZ	Amazon	1,879,572	1,162,941	4,955,492	Rating
DI	Discogs	1,754,823	270,771	5,302,276	Affiliation
FL	Flickr	395,979	103,631	8,545,307	Membership
DB	DBLP	1,953,085	5,624,219	12,282,059	Authorship
NY	NYTimes	299,752	101,636	69,679,427	Appearance
DE	Delicious	833,081	33,778,221	101,798,957	Interaction
OR	Orkut	2,783,196	8,730,857	327,037,487	Affiliation

- ooMBEA: the state-of-the-art algorithm proposed in (Chen et al., 2022) for enumerating all maximal bicliques.
- MSBE: our Algorithm 7 equipped with all the optimizations in Section 4.3.2.
- mat-MSBE: the materialized version of MSBE, as discussed at the end of Section 4.3.2.
- LG-MSBE and SS-MSBE: our index-based algorithms that use the largest gap and steady segment index, respectively.

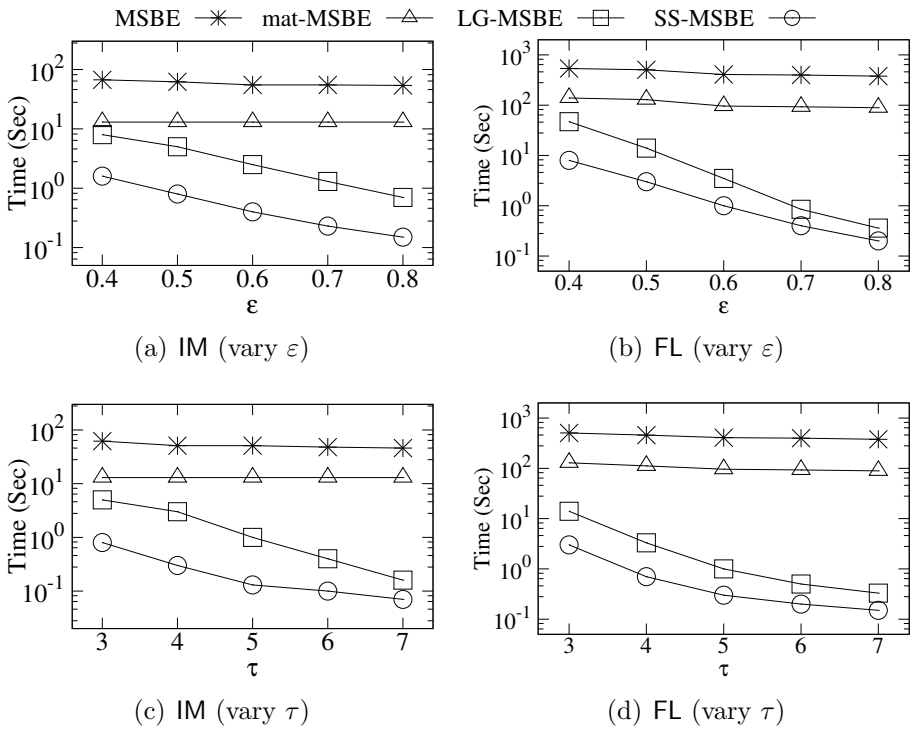
The source code of ooMBEA is obtained from the authors of (Chen et al., 2022).

All our algorithms are implemented in C++ and run in main memory. Without a further explanation, experiments are conducted on a machine with an Intel(R) 3.2GHz CPU and 64GB main memory running Ubuntu 18.04.5. We set a timeout of 10 hours for running an algorithm on a graph.

Datasets. We evaluate the algorithms on 17 real bipartite graphs, all of which are publicly available on KONECT [5]. Statistics of the graphs are shown in Table 4.1, where the graphs are listed in increasing order regarding the number of edges.

Query Parameters. A maximal similar-biclique enumeration query consists of two parameters, ε and τ. ε is chosen from $\{0.4, 0.5, 0.6, 0.7, 0.8\}$, and is set as 0.5 by default. τ is chosen from $\{3, 4, 5, 6, 7\}$, and is set as 3 by default. In addition, we also have parameters α and γ in index construction; we set $\alpha = 1$ and $\gamma = 0.3$ by default.

[5] http://konect.cc/networks/



(a) IM (vary ε) (b) FL (vary ε)

(c) IM (vary τ) (d) FL (vary τ)

Figure 4.10: Running time by varying ε and τ

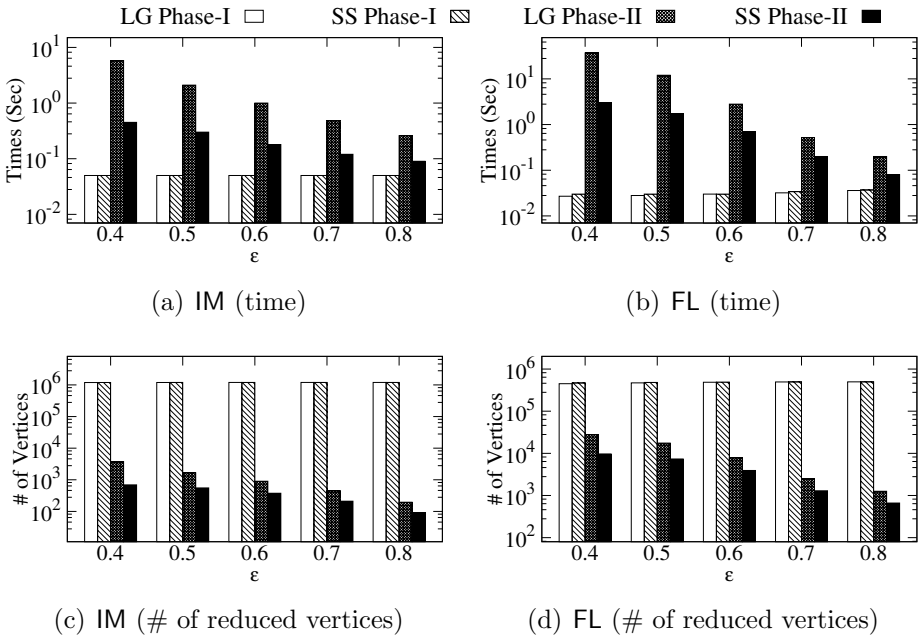
4.7.1 Efficiency Evaluations

In this subsection, we evaluate the efficiency of the algorithms. Note that, we also implemented a version of MSBE without the optimizations of Enum proposed in Section 4.3.2; it is omitted from the experiments since it times out in almost all the testings.

Running time on all graphs. The running time of the five algorithms on all graphs with default ε and τ is illustrated in Figure 4.9. We can see that mat-MSBE slightly improves upon MSBE when it is feasible to store the similar neighbors of all vertices in main memory. However, mat-MSBE runs out-of-memory on BS, CU, and DI, as marked by "oom" in Figure 4.9; for example, the memory consumption on BS would be over 400GB. Note that the memory consumption of mat-MSBE mainly depends on the structure, rather than the size, of the input graph, and thus mat-MSBE does not run out-of-memory on other larger graphs. Our two index-based algorithms, LG-MSBE and SS-MSBE, are the fastest and they outperform the other two algorithms that do not use index by up to 5 orders of magnitude. SS-MSBE is generally faster than LG-MSBE. Compared with the state-of-the-art maximal biclique enumeration algorithm ooMBEA, SS-MSBE is up to 6 orders of magnitude faster. Thus, we exclude ooMBEA from our remaining evaluations.

Running time by varying ε and τ. The running time of our four algorithms on IM and FL by varying ε and τ are shown in Figure 4.10. We can see that the running time of LG-MSBE and SS-MSBE decreases when either ε or τ increases. This is because, more vertices will be pruned by indexedVR when either ε or τ increases, and thus the enumeration process of LG-MSBE and SS-MSBE run faster.

Also, indexedVR runs faster when ε or τ increases, as can be seen from Figure 4.11. In contrast, the running time of MSBE and mat-MSBE is not so sensitive to ε or τ, as the dominating part of these two algorithms is computing similar neighbors for vertices.



(a) IM (time) (b) FL (time)

(c) IM (# of reduced vertices) (d) FL (# of reduced vertices)

Figure 4.11: Efficiency of indexedVR ($\tau = 3$)

Efficiency of indexedVR. In this experiment, we evaluate the efficiency of indexedVR for our two index structures. Recall that indexedVR (Algorithm 11) has two phases. Thus, we separately report the results of each phase. The running time on IM and FL are shown in Figures 4.11(a) and 4.11(b). We can see that the two index structures take almost the same time for the first phase, while the second phase of SS index-based indexedVR is much faster than LG index-based. This can partially be explained by the number of vertices that need to be pruned in the second phase, as reported in Figures 4.11(c) and 4.11(d). We remark that, for a fixed ε and τ, the total number of pruned vertices by different indexes are the same, and also the same as that pruned by the index-free approach VReduce. Thus, from the number of vertices that are pruned in Phase-II as shown in Figure 4.11, we can conclude that SS index prunes much more vertices than LG index in Phase-I. For example, for dataset FL and $\varepsilon = 0.4$, SS index prunes $467,329$ vertices in Phase-I and $9,691$ vertices in Phase-II, while LG index prunes $449,195$ vertices in Phase-I and $27,825$ vertices in Phase-II. As the second phase dominates the running time, SS index is superior.

Evaluate the optimizations for Enum. In this experiment, we evaluate the performance of our two optimization techniques (i.e., Lemma 7 and Lemma 8) on the Enum procedure. The results are reported in Figure 4.12. Specifically, we use our fastest algorithm SS-MSBE to evaluate these two optimization techniques. Note that SS-MSBE applies both optimizations. For this testing, we also implement SS-MSBE-v1 that adopts neither of the two optimizations and SS-MSBE-v2 that

Table 4.2: Index size and construction time

Graph	Size	Largest gap index		Steady segment index			
		consLG (s)	Size	consSS (s)	consSS* (s)	consSS** (s)	Size
YT	$4.6M$	3	$6.7M$	21	8	6	$8.3M$
GH	$7M$	1.1	$6.6M$	90	4.9	1.5	$6.2M$
LX	$9.2M$	48	$31M$	9,485	588	99	$50M$
BS	$12.4M$	506	$75M$	49,685	6,975	805	$59M$
BC	$18M$	16	$27M$	1,200	130	28	$30M$
AM	$24M$	1.5	$24M$	81	7	2.1	$20M$
WU	$30M$	11	$15M$	656	40	15	$16M$
CU	$36M$	1,130	$73M$	296,094	12,547	1,510	$103M$
TV	$50M$	11	$4.4M$	110	14	12	$2.3M$
IM	$58M$	10	$56M$	420	28	13	$56M$
AZ	$76M$	22	$122M$	2,220	152	37	$124M$
DI	$82M$	549	$132M$	45,967	5,187	863	$146M$
FL	$132M$	106	$30M$	3,102	235	122	$23M$
DB	$188M$	29	$299M$	1,905	177	45	$324M$
NY	$1.1G$	2,623	$35M$	20,934	4,397	2,708	$14M$
DE	$1.5G$	3,071	$2.3G$	129,304	13,435	3,910	$3.1G$
OR	$5G$	21,874	$690M$	246,045	23,872	23,164	$459M$

adopts only early termination (i.e., Lemma 7). From Figure 4.12 we can see that SS-MSBE-v1 cannot finish in a reasonable time due to lacking of these optimization techniques. SS-MSBE-v2 finishes successfully, while SS-MSBE is the most efficient algorithm on all datasets. This demonstrates that both of these two optimizations for Enum contribute to the efficiency.

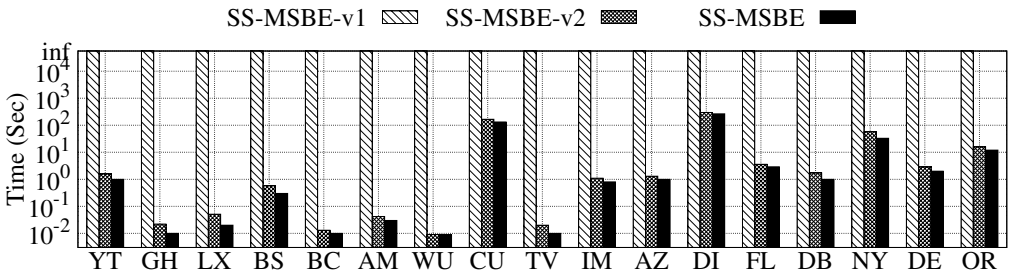


Figure 4.12: Evaluation of the optimizations for Enum

Index size and construction time on all graphs. The size of the two indexes on all graphs are shown in the fourth column and last column of Table 4.2. As a comparison, we also report the graph size in the second column of Table 4.2. We can see that in most cases, the sizes of the two indexes are similar to each other and are at the same level as the graph size, and thus they are affordable to be stored in main memory.

The running time of our index construction algorithms consLG, consSS, and consSS* are reported in the third, fifth, and sixth columns of Table 4.2, respectively. consLG runs the fastest due to its simplicity. Nevertheless, consSS*, which optimizes consSS by the similarity tree data structure, is only slightly slower than consLG.

Index performance by varying α. In this experiment, we evaluate the effect of

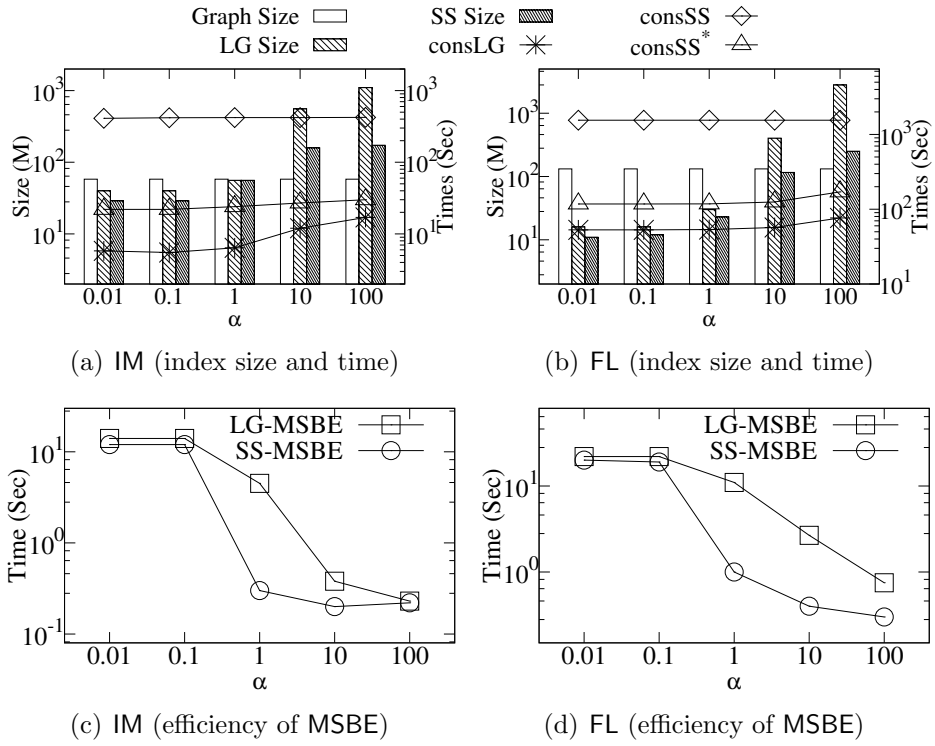


Figure 4.13: Index performance by varying α

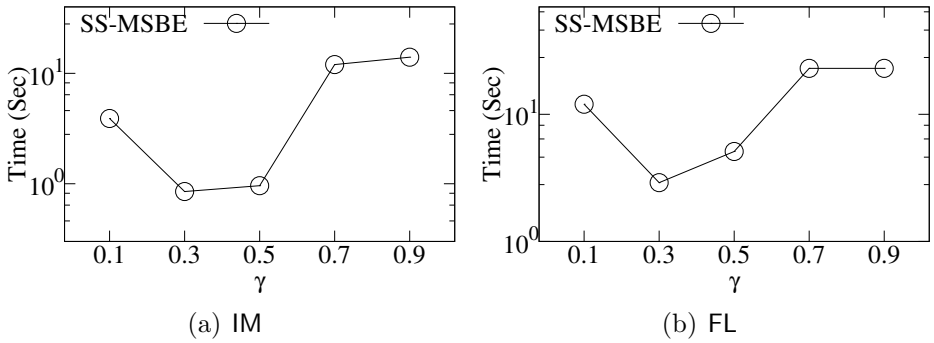


Figure 4.14: Efficiency of SS-MSBE by varying γ

α on the index size, index construction time and efficiency of MSBE. The results are shown in Figure 4.13. Recall that α controls the number of segments constructed for Φ_u. As expected, the index size and index construction time increase along with the increasing of α, as shown in Figures 4.13(a) and 4.13(b). When α is no larger than 1, the index size is at most at the same level as the graph size, but when α reaches 100, the index size can be much larger than the graph size. As shown in Figures 4.13(c) and 4.13(d), the running time of both LG-MSBE and SS-MSBE decreases when α increases. This is because the more the number of segments, the fewer the number of fake vertices. To strike a balance between index size and efficiency of MSBE, we recommend to set $\alpha \in [0.1, 10]$.

Efficiency of SS-MSBE by varying γ. In this experiment, we evaluate the performance of SS-MSBE for different γ values. Note that, the index size and index

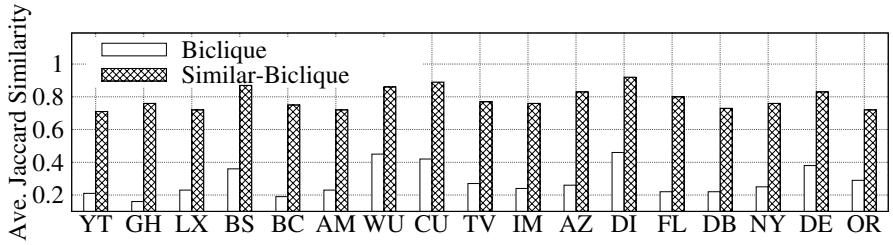
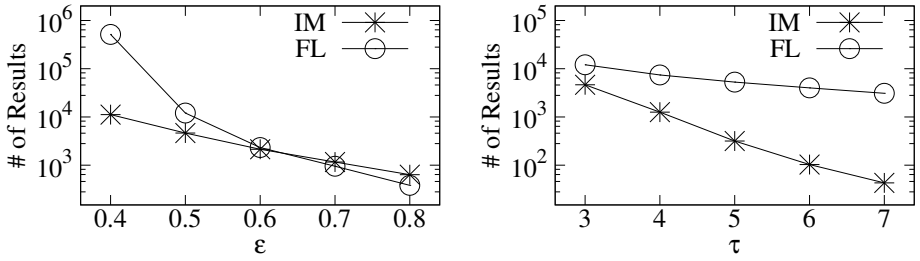


Figure 4.15: Average Jaccard similarity



(a) Varying ε (b) Varying τ

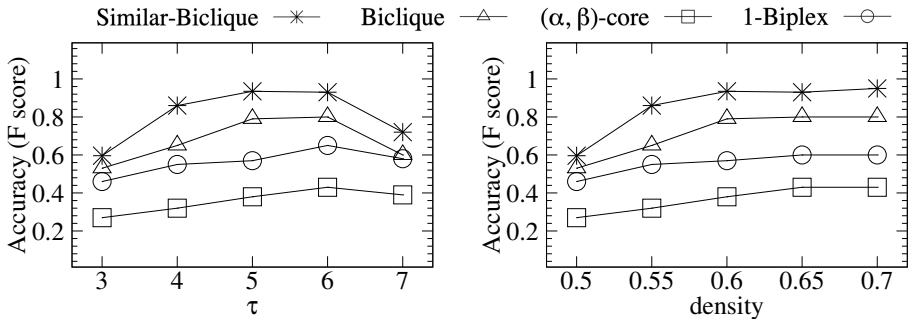
Figure 4.16: Number of maximal similar-bicliques

construction time of consSS are almost not affected by γ; thus we omit these results. This is because consSS selects a fixed number of steady segments (i.e., $\alpha \log |\Phi_u|$) to cover as many vertices of Φ_u as possible, and then it covers all remaining uncovered vertices of Φ_u by using the fewest number of disjoint segments. Thus, the total number of segments generated for Φ_u is at most $2\alpha \log |\Phi_u| + 1$, which is independent of γ. Figure 4.14 shows the running time of SS-MSBE by varying γ from 0.1 to 0.9. We can see that when γ is small (e.g., $\gamma < 0.3$), the performance of SS-MSBE is not good. The main reason is that when γ is small, a steady segment will cover fewer vertices due to the tighter constraint. As a result, more vertices need to be covered by the ordinary segments, which then results in introducing more fake vertices. Also, when γ is large, the performance of SS-MSBE becomes worse. This is because for large γ (e.g., $\gamma = 1$), a steady segment is no longer steady and degenerates to the ordinary segment. This motivates us to introduce steady segment. We recommend the value of γ to be in $[0.3, 0.5]$.

4.7.2 Effectiveness Evaluations

Average Jaccard similarity. We compare the average Jaccard similarity between L-side vertices in a maximal (similar-)biclique. Specifically, for each maximal (similar-)biclique C, we compute the average of the Jaccard similarity between all pairs of vertices from C_L, and then the average result of all maximal (similar-)bicliques is reported in Figure 4.15. We can see that vertices in a similar-biclique are much more similar to each other than in a biclique.

Number of maximal similar-bicliques. The number of maximal similar-bicliques for different ε and τ values are shown in Figure 4.16. We can see that the number of maximal similar-bicliques decreases with the increase of either ε or τ, which is



(a) Varying τ (b) Varying injected subgraph density

Figure 4.17: Case study 1: anomaly detection

as expected. This is because the number of similar neighbors for a vertex decreases with the increase of ε. Thus, more and more similar-bicliques disappear under a larger ε. It is worth mentioning that even under a high ε value of 0.7, there is still quite a few similar-bicliques, which exhibit a high level of consistence among members from the same side.

Case study 1: anomaly detection. We compare similar-biclique with other dense bipartite subgraph models, biclique, (α, β)-core (Kumar et al., 1999) and k-biplex (Yu et al., 2021), on anomaly detection in e-commerce applications. As mentioned in the Introduction, to improve the ranking of certain products, e-business owners may employ a set of fraudulent users to purchase a set of designated products. The fraudsters will also purchase other honest products trying to look "normal"; this is called "camouflage" in the literature. We consider a camouflage attack in the same way as (Hooi et al., 2016) on "Amazon Review Data" (Magazine Subscriptions) [6], which contains 65,546 reviews on 2,316 magazines by 53,617 users, by injecting 100 fraudulent users and 100 fraudulent products with various edge densities. The amount of camouflage (i.e., edges linking to honest products) added per fraudulent user is equal to the amount of fraudulent edges for that user. We adopt F-score, $\frac{2\times\text{precision}\times\text{recall}}{\text{precision}+\text{recall}}$, to evaluate the accuracy of detecting suspicious users and products. We apply the size constraint τ to all the models, where $\alpha = \beta = \tau$ for the (α, β)-core model; for our similar-biclique model, ε is set as 0.2. The results by varying τ and varying the density of the injected subgraph are shown in Figure 4.17. We can see that similar-biclique always achieves the highest accuracy. This is due to the similarity constraint imposed on users by similar-biclique, which naturally captures the reality that fraudulent users usually display a high level of synchronized behavior with each other. In contrast, biclique, 1-biplex, and (α, β)-core all have a low precision and thus low F-score. It is worth noting that the accuracy exhibits a peculiar trend where it increases initially and then decreases as τ ranges from 3 to 7. This phenomenon can be attributed to the detection of numerous trivial similar-bicliques with small sizes at lower τ values, such as $\tau = 3$, which ultimately results in a reduction of accuracy. Conversely, when τ is set to a larger value, such as $\tau = 7$, only a limited number of similar-bicliques are detected due to the size constraint, leading to a decrease in accuracy.

[6] https://nijianmo.github.io/amazon/index.html

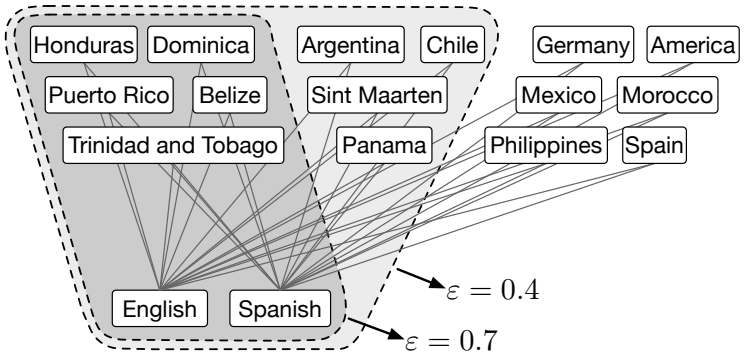


Figure 4.18: Case study 2: similar-bicliques in Unicode ($\tau = 2$)

Table 4.3: Case study 3: similar-bicliques in DBLP

Researchers	Research Groups
Albert Reuther, Andrew Prout, Antonio Rosa, Bill Bergeron, Chansup Byun, David Bestor, Julie Mullen, Matthew Hubbell, Peter Michaleas, William Arcand	High performance computing@MIT Lincoln Laboratory
Christian Menolfi, Lukas Kull, Marcel A. Kossel, Matthias Braendli, Pier Andrea Francese, Thomas Morf	CMOS integrated circuits@IBM Research–Zurich
Calvin Yu-Chian Chen, Chang-Hai Tsai, Chien-Yu Chen 0002, Da-Tian Bau, Fuu-Jen Tsai, Hung-Jin Huang, Ming-Hsui Tsai, Tin-Yun Ho, Yea-Huey Chang, Yuan-Man Hsu	Molecular biophysics @Taiwan
.....	.....

Case study 2: interesting pattern detection on Unicode. We also conduct a case study on the Unicode dataset (Kunegis, 2013) to illustrate the hierarchical structure of similar-bicliques by varying the similarity threshold ε. Unicode captures the languages that are spoken in a country. The three similar-bicliques detected for $\varepsilon = 0.7, 0.4, 0.01$ are reported in Figure 4.18, where the entire result corresponds to $\varepsilon = 0.01$; the similarity constraint is imposed on the countries and $\tau = 2$. We have the following observations. Firstly, the five countries in the similar-biclique for $\varepsilon = 0.7$ are all located in the Caribbean Sea Area with English and Spanish being their main language (around 90% population speak English and Spanish). Secondly, more countries from Latin America, e.g. Argentina and Chile, are included in the similar-biclique for $\varepsilon = 0.4$, and the newly added four countries speak more diverse languages. For example, in Sint Maarten, besides English and Spanish, around 8% population speak Virgin Islands Creole English and 4% population speak Dutch [7]. Lastly, when ε is 0.01, similar-biclique degenerates to biclique, and more countries are included, e.g., America and Germany. This demonstrates that similar-biclique can detect interesting patterns.

Case study 3: research group identification in DBLP. Our similar-biclique model also supports the case that not all vertices in a side share a common neighbor. In this case study, we show the similar-bicliques in a researcher-write-paper bipartite graph DBLP [8] by using different size constraints τ_L and τ_R on the two sides. The results for $\varepsilon = 0.6$, $\tau_L = 6$ and $\tau_R = 0$ are illustrated in Table 4.3; thus, researchers

[7] https://www.unicode.org/cldr/cldr-aux/charts/25/summary/root.html
[8] https://dblp.uni-trier.de/xml/

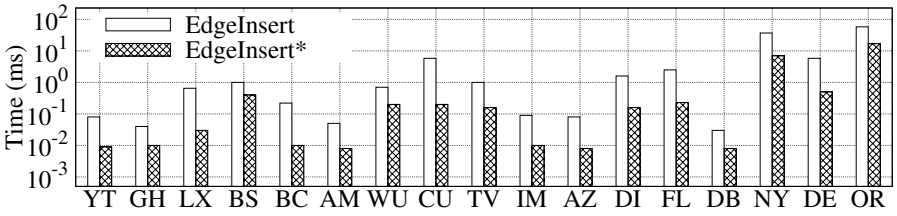


Figure 4.19: Running time for edge insertion

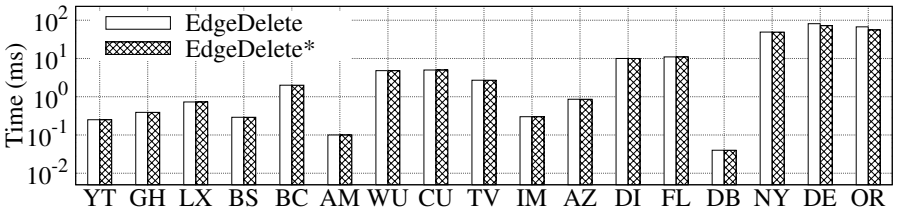


Figure 4.20: Running time for edge deletion

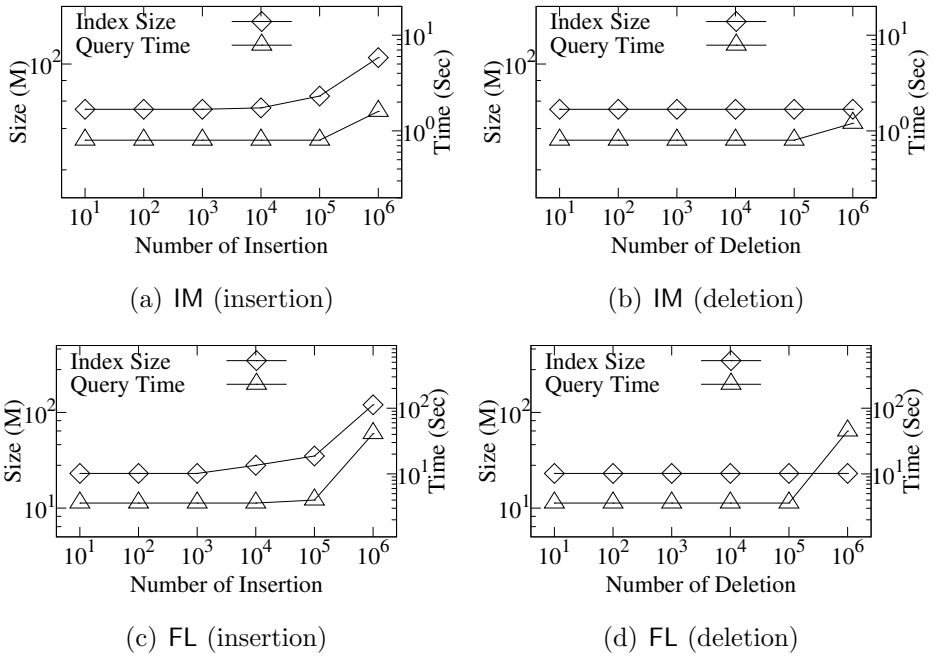
in a similar-biclique are not necessarily co-authors of the same paper. We find that the detected similar-bicliques corresponds to research groups in different institutes.

4.7.3 Index Maintenance Evaluations

Efficiency of EdgeInsert* and EdgeDelete*. In this experiment, we evaluate the performance of our index maintenance algorithms EdgeInsert* and EdgeDelete*. For comparison, we also report the performance of EdgeInsert and EdgeDelete, which do not apply the optimization techniques, i.e., Lemma 11 and Lemma 12. For edge insertion evaluation, we randomly insert $1,000$ new edges into the graph one by one and report the average processing time. For edge deletion evaluation, we randomly delete $1,000$ current edges from the graph one by one and report the average processing time. As shown in Figure 4.19, the running time of EdgeInsert* is at millisecond-level. For example, when adding a new edge into DB, the maintenance of the index can be completed less than 0.01 ms. Besides, we can see EdgeInsert* is faster than EdgeInsert by 1 order of magnitude on almost all datasets, which demonstrates the effectiveness of our optimization technique. We also evaluate the efficiency of EdgeDelete* in Figure 4.20. We can see the running time of EdgeDelete* is also at millisecond-level. An interesting phenomenon is that the effectiveness of optimization technique used in EdgeDelete* is as obvious as the one used in EdgeInsert*. The reason is that, for two random vertices from different sides, they are less likely to be connected due to the sparsity of the bipartite graphs in the real-world. Thus, the pruning power of Lemma 12 is not as strong as Lemma 11.

Index performance by increasing the number of updates. In this experiment, we evaluate the performance of the index by increasing the number of updates. Specifically, we evaluate the index size and query time by inserting more and more new edges into the graph, as well as deleting more and more current edges from the graph. Here, the query time is the running time of SS-MSBE with default setting

(i.e., $\varepsilon = 0.5$ and $\tau = 3$). As shown in Figure 4.21(a), the index size and query time start increasing when the number of inserted edges is over 10^4. This is because more segments will be created in our index and SS-MSBE will spend more time on these new created segments. Figure 4.21(b) reports the case of edge deletion. We can see the index size keeps steady since deleting edges will not bring new segments to our index. The query time increase when the number of updates is over 10^5. This is because the number of results (i.e., maximal similar-bicliques) increases significantly in this case, which needs more computation. Similar phenomenon can also be observed on other datasets. In this experiment, we observed that the index size and query time deteriorate when the number of updates reaches the scale of the number of edges in the graph. For example, the number of edges of IM is of million-scale (i.e., 3.7 million). The index performance deteriorates when the number of updates reaches 1 million as shown in Figure 4.21(a) and Figure 4.21(b). This experiment demonstrates the strategy to rebuild the index from scratch is reasonable and we recommend the value of ζ to be less than 1.



(a) IM (insertion) (b) IM (deletion)

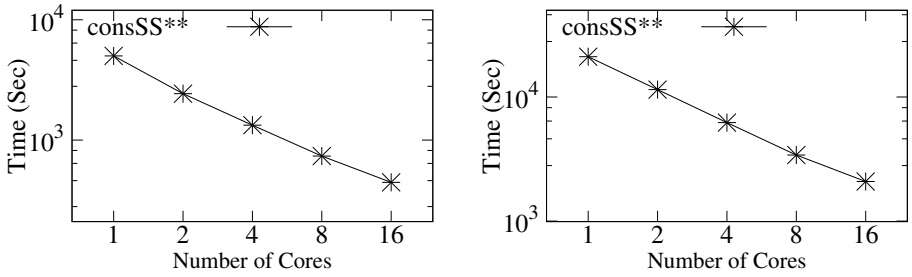
(c) FL (insertion) (d) FL (deletion)

Figure 4.21: Index performance w.r.t the number of updates

4.7.4 Parallel Algorithm Evaluations

In this subsection, we evaluate the parallelized version of algorithm consSS** by varying the number of CPU cores from 1 to 16 on our largest datasets DE and OR. This experiment was conducted on a machine with an Intel(R) 2.6GHz CPU and 16 cores. As shown in Figure 4.22, consSS** has a near-linear speedup by using multiple CPU cores. For example, on OR, the running time of consSS** decreases from 21,101 seconds (around 6 hours) to 2,084 seconds (around 0.5 hour) when we increase the core number from 1 to 16. This experiment demonstrates that the index

construction can be easily parallelized. Our index can be constructed more efficiently with the help of multiple CPU cores, which makes our index more attractive.



(a) DE (b) OR

Figure 4.22: Speeding up consSS** by using multi-core

4.8 Chapter Summary

In this chapter, we formulated the notion of similar-biclique, and proposed algorithms as well as optimization techniques to enumerate all similar-bicliques in a bipartite graph. Besides, index structures are also designed to speed up the computation. We also proposed effective and efficient index construction algorithms by investigating two different strategies. In addition, we proposed index maintenance algorithms to handle dynamic graph updates. Finally, we parallelized our index construction algorithms. Extensive empirical studies on real bipartite graphs demonstrated the effectiveness of our similar-biclique model and the efficiency of our algorithms. Case studies show that the similar-biclique model can be used to detect anomalies as well as interesting dense subgraph patterns. Our work initiates the study of integrating similarity constraint into dense bipartite subgraph mining, by taking the biclique model. For future studies, it will be interesting to integrate similarity constraints into other dense bipartite subgraph models, such as quasi-biclique, k-biplex, (α, β)-core, k-bitruss and k-wing. We believe that our proposed index structures will also be useful for these extensions.

Chapter 5

Structural Balanced Clique Identification

In this chapter, we study the structural balanced clique identification problem in signed graphs. The work is published in (Yao et al., 2022a). This chapter is organized as follows. Section 5.1 provides the introduction to this work. Section 5.2 provides the preliminaries. Section 5.3 studies the maximum balanced clique computation problem. Section 5.4 studies the large balanced clique enumeration problem. Section 5.5 studies the polarization factor computation problem. Section 5.6 studies the generalized maximum balanced clique problem. Experimental results are reported in Section 5.7. Finally, Section 5.8 concludes the chapter.

5.1 Introduction

Signed graphs enhance the representation capability of traditional graphs, by capturing the *polarity* of relationships between entities/vertices through *positive* and *negative* edge signs (Tang et al., 2016b). For example, signed graphs capture the friend-foe relationship in social networks (Easley and Kleinberg, 2010), support-dissent opinions in opinion networks (Kunegis et al., 2009), trust-distrust relationship in trust networks (Giatsidis et al., 2014), and activation-inhibition in protein-protein interaction networks (Ou-Yang et al., 2015). One prominent and fundamental theory in signed graph analysis is the *structural balance theory* (Harary et al., 1953), which states that a signed (sub)graph is structural balanced if its vertices can be partitioned into two sets such that all edges inside each partition have positive signs and all cross-partition edges have negative signs. That is, "the friend of my friend is my friend", and "the friend of my enemy is my enemy". Many interesting problems, such as community detection (Chu et al., 2016; Ordozgoiti et al., 2020), link prediction (Leskovec et al., 2010; Ye et al., 2013) and recommendation systems (Chen et al., 2013; Tang et al., 2016a), have been formulated and studied for signed graphs based on the structural balance theory.

Recently, the problem of enumerating all maximal structural balanced cliques in a signed graph is formulated and studied in (Chen et al., 2020). A vertex set C is a structural balanced clique if (1) it is a clique (i.e., every pair of its vertices is connected by an edge), and (2) it is structural balanced according to the structural

balance theory (i.e., C can be partitioned into two sets C_L and C_R such that all negative edges are between C_L and C_R). For presentation simplicity, we refer to structural balanced cliques as *balanced cliques*. For example, for the toy signed graph in Figure 5.1 that captures the sentiment among different subreddits on Reddit, the three "Apple" groups (in red) and the two "Android" groups (in blue) together form a balanced clique; here, solid (resp. dashed) lines represent positive (resp. negative) sentiment. However, signed graphs may have an enormous number of maximal balanced cliques, of varying sizes. For example, the Douban dataset used in our experiments has more than 10^9 maximal balanced cliques. Enumerating all of them may overwhelm end-users. To remedy this, a threshold τ is adopted in (Chen et al., 2020) such that only maximal balanced cliques C satisfying $|C_L| \geq \tau$ and $|C_R| \geq \tau$ are enumerated. Nevertheless, the number of such cliques could still be large, and the efficiency of the algorithms in (Chen et al., 2020) is not satisfactory.

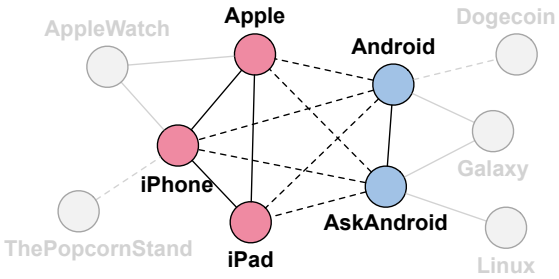


Figure 5.1: A toy signed graph of Reddit

Motivated by this, we in this chapter investigate both the *maximum* balanced clique computation problem and the *large* balanced clique enumeration problem. Given a signed graph $G = (V, E^+, E^-)$ and a threshold τ, the maximum balanced clique computation problem aims to find the largest balanced clique C^* in G that satisfies $|C_L^*| \geq \tau$ and $|C_R^*| \geq \tau$. Let $\omega_\tau(G)$ be the size of C^*, i.e., $\omega_\tau(G) = |C^*|$. Given G and two thresholds τ and α, the large balanced clique enumeration problem aims to enumerate all maximal balanced cliques $C \subseteq V$ that satisfy $|C_L| \geq \tau$, $|C_R| \geq \tau$ and $|C| \geq \omega_\tau(G) - \alpha$. By tuning α, end-users can make a trade-off between the size and the number of identified balanced cliques: the larger the value of α, the smaller the size and the more the number of reported balanced cliques. For example, by setting $\alpha = 0$, only *maximum* balanced cliques will be reported; by setting $\alpha = |V|$, all maximal balanced cliques will be reported. Note that, the maximum balanced clique in a signed graph is not unique. The maximum balanced clique computation problem reports an arbitrary maximum balanced clique, while the large balanced clique enumeration problem with $\alpha = 0$ reports all maximum balanced cliques. Detecting large balanced cliques has many applications.

- **Conflict Discovery.** Users actively interact with each other (both positively and negatively) on online platforms such as Facebook and Reddit, and these interactions can be modeled as a signed graph (Bonchi et al., 2019; Kumar et al., 2018; Xiao et al., 2020). Users in a large balanced clique are actively involved in conflicting groups, who have clear and firm standpoints on each other. Thus, they may represent core members of two polarized structures,

and detecting actively involved core members could help discover and prevent potential conflicts on the web.

- **Protein Complexes Detection.** Protein-protein interaction (PPI) networks can be modeled as signed networks to capture the activation-inhibition relations between proteins (Suratanee et al., 2014; Yim et al., 2018). As argued in (Ou-Yang et al., 2015; Vinayagam et al., 2014), protein complexes are ideally defined as groups of proteins within which are densely positively interacted (i.e., activation), and between which are densely negatively interacted (i.e., inhibition). Thus, detecting balanced cliques can help find protein complexes in signed PPI networks.

- **Synonym and Antonym Groups Discovery.** The synonyms and antonyms relationships between words can be naturally captured by a signed graph (Miller, 1995). Thus, we can use large balanced cliques to identify significant synonym groups that are antonymous with each other, which can be further used in applications such as semantic expansion (Krishnan et al., 2018) and automatic question generation (Kumar et al., 2019).

We propose an efficient branch-and-bound algorithm MBC* to compute a maximum balanced clique C^*. MBC* is based on the following observation. Suppose we know that C^* contains a vertex u, i.e., $u \in C^*$. Then, according to the structural balance theory, it must satisfy $C_L^* \subseteq \{u\} \cup N_G^+(u)$ and $C_R^* \subseteq N_G^-(u)$, and therefore C^* can be found in the subgraph G_u of G induced by $\{u\} \cup N_G^+(u) \cup N_G^-(u) = \{u\} \cup N_G(u)$; here, $N_G^+(u)$ (resp. $N_G^-(u)$) is the set of positive (resp. negative) neighbors of u in G. Moreover, based on the information of C_L^* and C_R^*, we can sparsify G_u by removing all *conflicting edges*: negative edges between vertices of $N_G^+(u)$, negative edges between vertices of $N_G^-(u)$, and positive edges between a vertex of $N_G^+(u)$ and a vertex of $N_G^-(u)$. In addition, after removing the conflicting edges from G_u, we no longer need to explicitly consider the structural balance theory and thus edge signs, as every clique of G_u will now be structural balanced. Let g_u be the resulting graph of G_u by discarding edge signs, and define $V_L = \{u\} \cup N_G^+(u)$ and $V_R = N_G^-(u)$. We call a clique in g_u that has at least τ vertices from each of V_L and V_R as a *dichromatic clique*. We show that C^* must be a maximum dichromatic clique in g_u that includes u. However, we do not know which vertex is in C^*. Thus, we need to enumerate each vertex u of G and suppose that u is in C^*. That is, we transform the maximum balanced clique computation problem over G to a series of maximum dichromatic clique computation problems over small subgraphs of G. As a result of the small size and no edge signs in g_u, the maximum dichromatic clique containing u in g_u can be efficiently computed by exploiting the existing pruning and bounding techniques that are originally designed for the classic maximum clique problem on unsigned graphs. Also, our empirical study shows that most, if not all, of the instances of maximum dichromatic clique computation problem are directly pruned by the bounding techniques.

In our problem formulation of enumerating large maximal balanced cliques, we require the end-user to specify a relative size threshold α for C (i.e., $|C| \geq \omega_\tau(G) - \alpha$), instead of an absolute threshold λ (i.e., $|C| \geq \lambda$). This is because a relative size

threshold α is much easier to specify than an absolute threshold λ: a too large λ may lead to no results, while a too small λ may lead to an enormous number of results. We propose a two-stage approach BCE* for enumerating large maximal balanced cliques. We first compute the maximum balanced clique size $\omega_\tau(G)$ by invoking MBC* in Stage-I, and then enumerate all large maximal balanced cliques by using the size threshold $\lambda = \max\{\omega_\tau(G) - \alpha, 2\tau\}$ in Stage-II. Our enumeration algorithm follows a similar idea to MBC*, i.e., we transform the enumeration problem over G into a series of enumeration problems over dichromatic-networks g_u for $u \in V(G)$. It is worth mentioning that when $\alpha \geq \omega_\tau(G) - 2\tau$, our problem becomes the same as the maximal balanced clique enumeration problem studied in (Chen et al., 2020). Our empirical studies show that BCE* is up to two orders of magnitude faster than the algorithm of (Chen et al., 2020) for this special case, i.e., for large α values. Note also that, our algorithm BCE* will run faster when α become smaller, while the algorithm of (Chen et al., 2020) does not consider α.

Both our maximum balanced clique problem and the maximal balanced clique enumeration problem studied in (Chen et al., 2020) require a user-given threshold τ. However, it is unclear how to choose the appropriate τ for an application. Choosing a too large τ may lead to no result, while a too small τ may lead to skewed results as well as an enormous number of results for the enumeration problem of (Chen et al., 2020). We provide two alternative ways to resolve this issue. Firstly, we formulate the *polarization factor problem*, which computes the largest τ^* such that G has a balanced clique C satisfying $|C_L| \geq \tau^*$ and $|C_R| \geq \tau^*$. We call this τ^* the *polarization factor* of G, denoted $\beta(G)$. It is immediate that there is no balanced clique for $\tau > \beta(G)$. Our empirical study shows that $\beta(G)$ varies from 3 to 201 for the graphs tested in our experiments. Thus, it would be interesting to know the polarization factor of a graph. We show that we can directly adapt our techniques of MBC* for computing $\beta(G)$. Secondly, instead of requiring users to input τ, we report a maximum balanced clique for every $\tau \geq 0$; we term this problem as the *generalized maximum balanced clique problem*. It is easy to see that the maximum balanced clique for τ must be no smaller than that for $\tau+1$, since an optimal solution to the latter is a feasible solution to the former. Thus, we vary τ from $\beta(G)$ to 0, and use the optimal solution to the problem for $\tau + 1$ as an initial solution to the problem for τ, for the purpose of computation sharing.

Our main contributions are summarized as follows.

- We propose an efficient branch-and-bound algorithm MBC* to solve the maximum balanced clique computation problem. Our main idea is based on a novel graph reduction technique that transforms the problem over a large signed graph G to a series of maximum dichromatic clique computation problems over small subgraphs of G, which not only removes edge signs but also sparsifies the edge set.

- We formulate the large balanced clique enumeration problem for a relative size threshold α, which captures the maximal balanced clique enumeration problem studied in (Chen et al., 2020) as a special case. We also propose an efficient algorithm BCE* to enumerate all large maximal balanced cliques.

- We formulate the polarization factor problem, and modify MBC* to efficiently

solve the polarization factor problem. The polarization factor $\beta(G)$ provides a rough guidance for choosing the threshold τ for our maximum balanced clique problem as well as the maximal balanced clique enumeration problem of (Chen et al., 2020).

- We also extend our techniques for the generalized maximum balanced clique problem that reports a maximum balanced clique for each $\tau \geq 0$.
- We conduct extensive empirical studies on large real signed graphs to demonstrate the effectiveness of our models and the efficiency of our algorithms. In particular, for the problem of enumerating all maximal balanced cliques that is studied in (Chen et al., 2020), our algorithm BCE* is up to two orders of magnitude faster than the algorithm proposed in (Chen et al., 2020).

5.2 Preliminaries

In this chapter, we focus on an *undirected* and *signed* graph $G = (V, E)$, where V is the set of vertices and E is the set of signed edges that is further partitioned into *positive* edges E^+ and *negative* edges E^-. We also denote a signed graph as $G = (V, E^+, E^-)$. We assume that the signed graph G is *simple*, i.e., $E^+ \cap E^- = \emptyset$ and there is no self-loops. We denote the number of vertices and the number of edges by n and m, respectively, i.e., $n = |V|$ and $m = |E| = |E^+| + |E^-|$. For each vertex $v \in V$, let $N_G^+(v) = \{u \mid (v, u) \in E^+\}$ be the set of *positive neighbors* of v and $N_G^-(v) = \{u \mid (v, u) \in E^-\}$ be the set of *negative neighbors* of v. We use $d_G^+(v) = |N_G^+(v)|$ and $d_G^-(v) = |N_G^-(v)|$ to denote the *positive degree* and *negative degree* of v, respectively. We also use $N_G(v)$ and $d_G(v)$ to denote the (entire set of) neighbors and (total) degree of v, i.e., $N_G(v) = N_G^+(v) \cup N_G^-(v)$ and $d_G(v) = |N_G(v)| = d_G^+(v) + d_G^-(v)$. For ease of presentation, we omit the subscript G in the notations when the context is clear. Given a vertex subset $S \subseteq V$, we use $G[S]$ to denote the *vertex-induced* subgraph of G that consists of all edges between vertices of S, i.e., $G[S] = (S, \{(u, v) \in E \mid u \in S, v \in S\})$.

Definition 9 (Structural Balanced Group (Harary et al., 1953))**.** Given a signed graph $G = (V, E^+, E^-)$, a group of vertices $C \subseteq V$ is *structural balanced* if it can be split into two subgroups C_L and C_R such that all edges between vertices in the same subgroup are positive and all edges between vertices from different subgroups are negative.

Definition 10 (Structural Balanced Clique (Chen et al., 2020))**.** Given a signed graph G, a group of vertices $C \subseteq V$ is a *structural balanced clique* if (1) it is structural balanced and (2) it induces a clique, i.e., $(u, v) \in E^+ \cup E^-, \forall u, v \in C$ with $u \neq v$.

Example 15. *Consider the signed graph in Figure 5.2 where positive (resp. negative) edges are represented by solid (resp. dashed) lines.* $C = \{v_1, v_2, v_3, v_4\}$ *is a structural balanced clique with* $C_L = \{v_1, v_2\}$ *and* $C_R = \{v_3, v_4\}$*, or* $C_L = \{v_3, v_4\}$ *and* $C_R = \{v_1, v_2\}$.

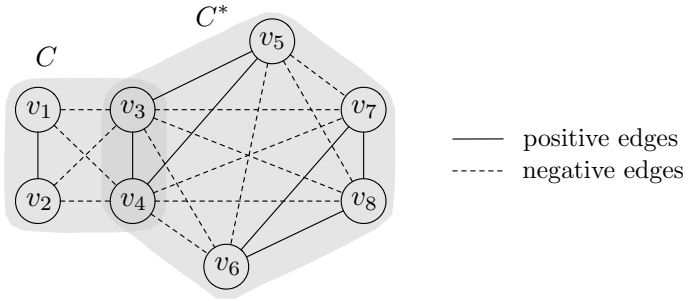


Figure 5.2: A toy signed graph

For simplicity, we refer to a structural balanced clique as a *balanced clique*. We call C_L and C_R the two sides (e.g., left and right) of the balanced clique C. It is easy to verify that the splitting of C into C_L and C_R is unique; nevertheless, the roles of C_L and C_R can be swapped. Thus, in the following, we directly use C_L and C_R to denote the two sides without formally defining them. Note that if all edges between vertices of C are positive edges, then C is also regarded as structural balanced. That is, one of C_L and C_R can be empty.

5.3 Maximum Balanced Clique Computation

In this section, we study the maximum balanced clique problem.

Problem 3 (Maximum Balanced Clique Problem)**.** Given a signed graph G and a non-negative polarization threshold τ, the maximum balanced clique problem aims to find the largest balanced clique C^* in G that satisfies the *polarization constraint* τ (i.e., $|C_L^*| \geq \tau$ and $|C_R^*| \geq \tau$).

In the problem definition, the polarization constraint τ requires each side (i.e., C_L and C_R) to be of size at least τ, in the same way as (Chen et al., 2020). This is to make sure that the sides are not extremely small, and to avoid reporting too skewed results. Consider the signed graph in Figure 5.2 and suppose $\tau = 2$. Both $C = \{v_1, v_2, v_3, v_4\}$ and $C^* = \{v_3, v_4, v_5, v_6, v_7, v_8\}$ (also some subsets of C^*) are balanced cliques and satisfy the polarization constraint τ, while C^* is the largest one.

Theorem 5. *The maximum balanced clique computation problem is NP-hard.*

Proof. We prove the NP-hardness of the problem by reducing from the classic maximum clique problem over unsigned graphs which is NP-hard (Karp, 1972). Given an unsigned graph instance $G = (V, E)$ of the maximum clique problem and any threshold $\tau \leq |V|$, we construct the signed graph instance G^s for the maximum balanced clique problem as follows. We first set G^s as G with all edges being positive edges. Then, we add into G^s a complete graph with τ vertices $\{u_1, \ldots, u_\tau\}$ where all edges are positive edges. Finally, we add into G^s a negative edge between each vertex of $\{u_1, \ldots, u_\tau\}$ and each vertex of V. It is easy to verify that G has a clique of size at least τ if and only if G^s has a balanced clique satisfying the polarization constraint τ. Thus, the maximum balanced clique computation problem is NP-hard. □

5.3.1 An Enumeration-based Baseline Approach MBC

The problem of enumerating all maximal balanced cliques has been studied in (Chen et al., 2020), and the enumeration algorithm MBCEnum proposed in (Chen et al., 2020) can be easily modified to find the maximum balanced clique. In the following, we first describe MBCEnum, and then present our enumeration-based baseline algorithm MBC for the maximum balanced clique problem.

MBCEnum is an adaptation of the classic BK algorithm, proposed in (Bron and Kerbosch, 1973) for enumerating all maximal cliques in unsigned graphs, to enumerating all maximal balanced cliques. The general idea is to iteratively build up two sets C_L and C_R such that $C_L \cup C_R$ is always a balanced clique. In addition, it maintains two candidate sets P_L and P_R of vertices that can be used to grow C_L and C_R, respectively. Specifically, P_L (resp. P_R) is the set of vertices that are directly connected to every vertex of C_L (resp. C_R) via positive edges and are directly connected to every vertex of C_R (resp. C_L) via negative edges. Then, it tries each vertex of P_L to be added to C_L and each vertex of P_R to be added to C_R, to grow the solution by one vertex and then conducts the recursion. Note that, MBCEnum also maintains two exclusive sets X_L and X_R of vertices that are used for certifying whether a solution $C_L \cup C_R$ is maximal or not; we do not discuss them here as they are not needed when computing the maximum balanced clique.

To improve the efficiency, MBCEnum (Chen et al., 2020) also proposed a vertex reduction method VertexReduction and an edge reduction method EdgeReduction to reduce the input graph based on the polarization threshold τ. The general idea of VertexReduction is that every vertex in a balanced clique satisfying the polarization constraint must have a positive degree at least $\tau - 1$ and a negative degree at least τ; thus, all vertices violating these degree constraints can be removed. The general idea of EdgeReduction is that every edge in a balanced clique satisfying the polarization constraint must participate in a certain number of triangles of each type; we omit the details, which can be found in (Chen et al., 2020). VertexReduction can be conducted in $\mathcal{O}(n+m)$ time, while EdgeReduction takes $\mathcal{O}(m^{3/2})$ time.

Based on MBCEnum, we have a baseline algorithm MBC for the maximum balanced clique problem. The pseudocode of MBC is shown in Algorithm 16, which is self-explanatory. It can be easily verified that all calls to $\mathsf{Enum}(C_L, C_R, P_L, P_R)$, except the first one, guarantee that $C_L \cap C_R = \emptyset$, $P_L \cap P_R = \emptyset$ and $(C_L \cup C_R) \cap (P_L \cup P_R) = \emptyset$. Note that at Line 11, we swap the roles of C_L (resp. P_L) and C_R (resp. P_R) such that we are adding vertices to the two sides of the growing balanced clique in alternating order; this is to avoid generating too skewed intermediate results.

5.3.2 A Maximum Dichromatic Clique-based Approach MBC*

Our empirical studies in Section 5.7 show that the MBC algorithm is inefficient, due to lack of advanced pruning and bounding techniques. That is, only size-based upper bound is used for pruning (see Line 10 of Algorithm 16). To improve efficiency, we aim to utilize (some of) the advanced pruning and bounding techniques that have been shown to be successful for the classic maximum clique problem in unsigned graphs (Chang, 2019, 2020; Maslov et al., 2014; Tomita, 2017). Specifically, we aim to utilize the *degree-based pruning* and *graph coloring-based upper bounding* for

Algorithm 16: MBC

Input: A signed graph $G = (V, E^+, E^-)$ and a threshold τ
Output: The maximum balanced clique C^*

Reduce G by VertexReduction and EdgeReduction of (Chen et al., 2020);
$C^* \leftarrow \emptyset$;
Enum$(\emptyset, \emptyset, V(G), V(G))$;
return C^*;

Procedure Enum(C_L, C_R, P_L, P_R)
if $|C_L| \geq \tau$ **and** $|C_R| \geq \tau$ **and** $|C_L| + |C_R| > |C^*|$ **then**
  $C^* \leftarrow C_L \cup C_R$;
for each $v \in P_L$ **do**
  $C'_L \leftarrow C_L \cup \{v\}$; $C'_R \leftarrow C_R$;
  $P'_L \leftarrow N^+(v) \cap P_L$; $P'_R \leftarrow N^-(v) \cap P_R$;
  if $|C'_L| + |P'_L| \geq \tau$ **and** $|C'_R| + |P'_R| \geq \tau$ **and** $|C'_L| + |P'_L| + |C'_R| + |P'_R| > |C^*|$ **then**
    if $P'_R \neq \emptyset$ **then** Enum(C'_R, C'_L, P'_R, P'_L);
    else Enum(C'_L, C'_R, P'_L, P'_R);
  $P_L \leftarrow P_L \setminus \{v\}$;

computing maximum balanced clique. To illustrate, let's consider an *unsigned* graph, and let lb be a lower bound of the maximum clique size which is set as the largest size of the enumerated cliques. The degree-based pruning is as follows.

Lemma 13 (Degree-based pruning)**.** *If the degree of a vertex u is less than lb, then we can remove u from the graph without affecting the maximum clique computation.*

Degree-based pruning is correct because we have already found a clique of size lb, and we are now searching for cliques of size larger than lb. A *coloring* of a graph is to assign a color to each vertex of the graph such that no two adjacent vertices have the same color. The smallest number of colors needed to color a graph is called its *chromatic number*.

Lemma 14 (Graph coloring-based upper bounding)**.** *The maximum clique size of a graph is at most its chromatic number.*

The correctness is easy to see as each vertex of a clique requires a different color. However, computing the chromatic number is an NP-hard problem (Karp, 1972). Thus, heuristic techniques (e.g., see (Chang, 2019)) are usually used in practice to compute an upper bound of the chromatic number, which then is also an upper bound of the maximum clique size.

Ineffectiveness of A Naive Strategy. However, it is nontrivial to effectively apply these pruning and bounding techniques to balanced clique computation on signed graphs. This is because we now have both positive edges and negative edges, and we also need to satisfy the structural balanced constraint. One possible way to utilize these techniques for signed graphs is ignoring the edge signs and the structural balanced constraint when conducting pruning and bounding. This is correct, but is

ineffective as verified by our experiments. First, the structural balanced constraint is not considered due to ignoring edge signs. Second, the number of edges is abundant which makes the pruning and bounding ineffective. Consider the signed graph in Figure 5.3 as an example. The number of colors needed to color its vertices after ignoring edge signs is 6 as there is an edge between every pair of vertices, and thus the coloring-based upper bound is 6. But it is easy to see that the maximum balanced clique size is 3 for $\tau = 0$, and is 2 for $\tau = 1$.

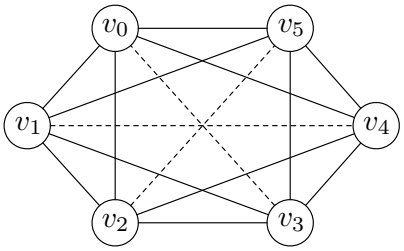


Figure 5.3: Ineffective of coloring-based upper bound by ignoring edge signs

A Novel Graph Reduction Technique. To resolve the above drawbacks, we propose a novel graph reduction technique. The general idea is based on the following observation that removes edges from a subgraph. Suppose we know that vertex u is in the maximum balanced clique C^*. Then, according to the structural balance theory, it must satisfy $C_L^* \subseteq \{u\} \cup N_G^+(u)$ and $C_R^* \subseteq N_G^-(u)$; here, without loss of generality, we assume $u \in C_L^*$. This is because by the structural balance theory, each vertex of $N_G^+(u)$ has a positive edge to u and thus cannot be on the opposite side of u, and similarly vertices of $N_G^-(u)$ cannot be on the same side as u. Therefore, C^* can be found in the subgraph G_u of G induced by vertices $\{u\} \cup N_G^+(u) \cup N_G^-(u)$. Moreover, based on the information of C_L^* and C_R^*, we can sparsify the subgraph G_u by removing all *conflicting edges*:

- negative edges between vertices of $N_G^+(u)$,
- negative edges between vertices of $N_G^-(u)$, and
- positive edges between a vertex of $N_G^+(u)$ and a vertex of $N_G^-(u)$.

This is because, if $v, v' \in N_G^+(u)$ are connected by a negative edge, then v and v' cannot be both in C^*; recall that $C_L^* \subseteq \{u\} \cup N_G^+(u)$. Similarly, if $v \in N_G^+(u)$ and $v'' \in N_G^-(u)$ are connected by a positive edge, then v and v'' cannot be both in C^*. As a result, all the conflicting edges are not in $G[C^*]$ and thus can be safely removed. In addition, after removing the conflicting edges from G_u, we no longer need to explicitly consider the structural balance theory and thus edge signs; this is because every clique of G_u will now be structural balanced in G. Let g_u be the resulting graph of G_u by discarding edge signs, and define $V_L = \{u\} \cup N_G^+(u)$ and $V_R = N_G^-(u)$. It is easy to verify that C^* is the maximum clique in g_u that includes u and has at least τ vertices from each of V_L and V_R. Thus, our problem becomes finding such a maximum clique in g_u; we term this problem as maximum dichromatic clique problem, which is formally defined as follows.

Problem 4 (Maximum Dichromatic Clique Problem)**.** The input of the maximum dichromatic clique problem consists of a *dichromatic graph* $g = (V(g), E(g))$ — where the vertices $V(g)$ is further partitioned into the set of L-vertices V_L and the set of R-vertices V_R with $V_L \cap V_R = \emptyset$ — and a non-negative threshold τ. It aims to find the largest clique $C^* \subseteq V(g)$ such that $|C^* \cap V_L| \geq \tau$ and $|C^* \cap V_R| \geq \tau$. We call a clique C satisfying $|C \cap V_L| \geq \tau$ and $|C \cap V_R| \geq \tau$ a *dichromatic clique* satisfying the constraint τ.

The above discussion is based on the assumption that we know a vertex u in the maximum balanced clique. However, in practice, we do not know such a vertex. To remedy this, we enumerate each vertex $u \in V(G)$ and compute a maximum balanced clique containing u; the largest one among all the enumerated balanced cliques then is the result. Actually, we can do better by giving a total ordering $\prec$ to the vertices, and only considering higher-ranked neighbors in constructing G_u and g_u. In summary, we transform the maximum balanced clique problem over a large signed graph G to a series of maximum dichromatic clique problems over small subgraphs of G, as follows. Given a signed graph $G = (V, E^+, E^-)$ and any total ordering $\prec$ of V, for each vertex $u \in V$,

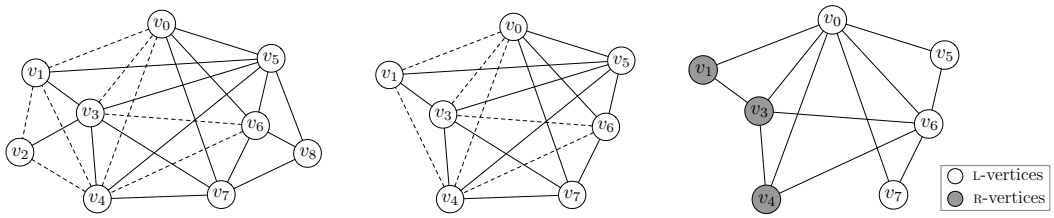
1. We first extract the *ego-network* of u, denoted G_u, which is the subgraph of G induced by u and u's *higher-ranked* neighbors according to the total ordering $\prec$.

2. We then transform G_u into a *dichromatic-network* of u, denoted g_u, which ***not only removes edge signs but also sparsifies the edge set***. Specifically, let V_L be the union of u and u's positive neighbors in G_u, and V_R be the set of u's negative neighbors in G_u. We label vertices of V_L as L-vertices, and vertices of V_R as R-vertices. Then, we remove all *conflicting edges* from G_u: all negative edges between L-vertices, all negative edges between R-vertices, and all positive edges between L-vertices and R-vertices. The resulting graph after discarding edge signs is a dichromatic graph g_u.

Example 16. *For example, consider the signed graph in Figure 5.4(a) and assume v_0 has the lowest rank according to the total ordering $\prec$. Figure 5.4(b) illustrates the ego-network G_{v_0} of v_0 which does not include vertices $\{v_2, v_8\}$, and Figure 5.4(c) shows the dichromatic-network g_{v_0} of v_0 which removes edges $\{(v_1, v_4), (v_1, v_5), (v_3, v_5), (v_4, v_5), (v_3, v_7), (v_4, v_7)\}$. Note that, in our implementation, we actually exclude u and its adjacent edges from G_u and g_u. Then, the effect of edge reduction becomes more evident; for example, G_{v_0} has 12 edges while g_{v_0} only has 6 edges, after excluding v_0.*

We prove in the theorem below that the maximum balanced clique in G can be obtained by computing maximum dichromatic cliques in the dichromatic networks g_u for all $u \in V$.

Theorem 6. *The size of the maximum balanced clique in $G = (V, E^+, E^-)$ that satisfies the constraint τ is equal to $\max_{u \in V} \delta(g_u, \tau)$, where $\delta(g_u, \tau)$ denotes the size of the maximum dichromatic clique in g_u satisfying the constraint τ.*



(a) A signed graph (b) Ego-network G_{v_0} (c) Dichromatic-network g_{v_0}

Figure 5.4: Illustration of our transformation

Proof. Let $\omega(G, \tau)$ be the size of the maximum balanced clique in G that satisfies the polarization constraint τ. First, for any dichromatic clique C in the dichromatic network g_u for $u \in V$, C is a balanced clique in G satisfying the polarization constraint τ. This is because, let $C_L = C \cap V_L(g_u)$ and $C_R = C \cap V_R(g_u)$, then all edges between C_L and all edges between C_R in g_u correspond to positive edges in the ego-network G_u, and all edges between C_L and C_R in g_u correspond to negative edges in G_u. Thus, $C_L \cup C_R$ is a balanced clique in G_u and thus in G, and $\omega(G, \tau) \geq \max_{u \in V} \delta(g_u, \tau)$. Second, let C^* be a maximum balanced clique in G satisfying the polarization constraint τ, and let u^* be the lowest-ranked vertex in C^*. Then C^* is a dichromatic clique in g_{u^*}, as C^*_L (that contains u^*) are L-vertices and C^*_R are R-vertices. Thus, $\omega(G, \tau) \leq \max_{u \in V} \delta(g_u, \tau)$, and the lemma holds. □

There are two main benefits of transforming the maximum balanced clique problem over G to a series of maximum dichromatic clique problems over dichromatic networks of G.

- Firstly, each dichromatic network is small as discussed above. Although we need to solve n instances of the maximum dichromatic clique problem in the worst case, our empirical studies in Section 5.7 show that we only need to solve a small number of such instances (e.g., at most hundreds) in practice. This is because most of the instances are directly pruned by the degree-based pruning and coloring-based upper bounding.

- Secondly, by transforming the maximum balanced clique searching on signed graphs to maximum dichromatic clique searching on dichromatic graphs, degree-based pruning and graph coloring-based upper bounding come into effect, which can significantly reduce the search space and thus speed up the computation.

Pseudocode of MBC*. Based on the above discussions, the pseudocode of our dichromatic clique-based algorithm for the maximum balanced clique problem is shown in Algorithm 17, denoted MBC*. We first apply the VertexReduction of (Chen et al., 2020) to reduce the input signed graph G (Line 1). Note that, we do not apply EdgeReduction of (Chen et al., 2020); this is because it has a high time complexity and incurs a large overhead for our efficient algorithm MBC*, as verified by our experiments in Section 5.7. Next, we heuristically compute a balanced clique by invoking MBC-Heu (Line 2), which will be presented in Section 5.3.3. Then, we

Algorithm 17: MBC*

Input: A signed graph $G = (V, E^+, E^-)$, and a threshold τ
Output: The maximum balanced clique C^*

1 Reduce G by VertexReduction of (Chen et al., 2020);
2 $C^* \leftarrow$ MBC-Heu(G, τ);
3 Reduce G to its $|C^*|$-core;
4 DOrder$(\cdot) \leftarrow$ degeneracy ordering of vertices;
5 **for each** *vertex u in reverse order w.r.t.* DOrder$(\cdot)$ **do**
6 $\quad g_u \leftarrow$ the dichromatic-network of u;
7 $\quad g \leftarrow$ the $|C^*|$-core of g_u;
8 $\quad$ **if** colorUB$(g) > |C^*|$ **then** MDC$(\{u\}, g \setminus \{u\}, \tau - 1, \tau)$;
9 **return** C^*;

Procedure MDC$(C, g = (V_L, V_R, E), \tau_L, \tau_R)$
10 **if** $|C| > |C^*|$ **and** $\tau_L \leq 0$ **and** $\tau_R \leq 0$ **then** $C^* \leftarrow C$;
11 Reduce g to its $(|C^*| - |C|)$-core;
12 **if** $|V_L(g)| < \tau_L$ **or** $|V_R(g)| < \tau_R$ **or** colorUB$(g) \leq |C^*| - |C|$ **then**
13 $\quad$ **return** ;
14 **if** $\tau_L > 0$ **and** $\tau_R \leq 0$ **then** $\mathcal{B} \leftarrow V_L(g)$;
15 **else if** $\tau_L \leq 0$ **and** $\tau_R > 0$ **then** $\mathcal{B} \leftarrow V_R(g)$;
16 **else** $\mathcal{B} \leftarrow V_L(g) \cup V_R(g)$;
17 **while** $\mathcal{B} \neq \emptyset$ **do**
18 $\quad v \leftarrow$ the vertex of $\mathcal{B}$ with the minimum degree in g;
19 $\quad$ **if** $v \in V_L(g)$ **then** $\tau'_L \leftarrow \tau_L - 1$; $\tau'_R \leftarrow \tau_R$;
20 $\quad$ **else** $\tau'_L \leftarrow \tau_L$; $\tau'_R \leftarrow \tau_R - 1$;
21 $\quad$ MDC$(C \cup \{v\}, g[N_g(v)], \tau'_L, \tau'_R)$;
22 $\quad$ Remove v from $\mathcal{B}$ and g;

reduce G to its $|C^*|$-core and obtain the degeneracy ordering of vertices (Lines 3–4), by ignoring edge signs. Here, the ***k-core*** is the maximal subgraph that has minimum degree at least k (Matula and Beck, 1983), which can be obtained by iteratively removing vertices of degree smaller than k. The ***degeneracy ordering*** (aka smallest-first ordering) is defined recursively as follows: the first vertex has the smallest degree in the graph, the second vertex has the smallest degree in the resulting graph after removing all preceding vertices, and so on (Matula and Beck, 1983). Based on the degeneracy ordering DOrder, we process vertices in the reverse order according to DOrder (Line 5); we say a vertex ranks higher if it appears later in DOrder. For each vertex u, we extract the dichromatic-network g_u of u (Line 6), and then compute the maximum dichromatic clique in g_u (Lines 7–8). The intuition of using the degeneracy ordering is that the ego-networks as well as the dichromatic networks then will have fewer vertices.

Efficiently computing the maximum dichromatic clique in the dichromatic network g_u is critical to the performance of our algorithm MBC*. We propose a branch-and-bound algorithm, MDC, to solve this problem efficiently, whose pseudocode is also given in Algorithm 17. The input of MDC consists of a clique C, a dichromatic graph $g = (V_L, V_R, E)$, and two thresholds τ_L and τ_R. It aims to find the largest

dichromatic clique C' in g such that $|C' \cap V_L| \geq \tau_L$ and $|C' \cap V_R| \geq \tau_R$; then $C \cup C'$ is used to update C^* (Line 10). Note that, the existing maximum clique solvers for unsigned graphs cannot be directly applied here due to the existence of the two thresholds τ_L and τ_R. We first reduce g to its $(|C^*|-|C|)$-core at Line 11 by ignoring vertex labels (i.e., L and R); this is because we are looking for a dichromatic clique C' such that $|C \cup C'| > |C^*|$. We prune the instance (i.e., g) if there is no feasible dichromatic clique (i.e., $|V_L(g)| < \tau_L$ or $|V_R(g)| < \tau_R$) or a computed coloring-based upper bound of the maximum clique size by ignoring vertex labels is no larger than $|C^*| - |C|$ (Lines 12-13). If the instance is not pruned, then we generate new branches. First, the branching vertices $\mathcal{B}$ are selected based on τ_L and τ_R as follows. If $\tau_L > 0$ and $\tau_R \leq 0$, the next vertex to be included into C (i.e., $\mathcal{B}$) should be ideally from $V_L(g)$ (Line 14). Similarly, if $\tau_L \leq 0$ and $\tau_R > 0$, the next vertex to be included into C should be from $V_R(g)$ (Line 15). Otherwise, we can choose any vertex from $V_L(g) \cup V_R(g)$ (Line 16). Then, while $\mathcal{B}$ is not empty (Line 17), we choose the vertex v of $\mathcal{B}$ that has the smallest degree in g (Line 18), to generate a new branch at Line 21 by including v into C and then recursively solve the problem on the subgraph of g induced by v's neighbors (i.e., $g[N_g(v)]$). After processing v, we remove v from both $\mathcal{B}$ and g (Line 22).

To reduce the number of MDC instances to be generated at Line 8 of Algorithm 17, we conduct the following prunings for g_u. We first reduce g_u to its $|C^*|$-core, denoted by g, by ignoring vertex labels (Line 7); this is because we are now searching for a clique of size at least $|C^*|+1$. Then, we compute a coloring-based upper bound for the maximum clique size of g after ignoring vertex labels. If this upper bound is no larger than $|C^*|$, then we prune the graph g without calling MDC (Line 8).

Theorem 7. *The time complexity of* MBC* *(Algorithm 17) is* $\mathcal{O}(n \cdot m \cdot 2^{\delta})$ *and the space complexity is* $\mathcal{O}(m)$*, where* δ *is the degeneracy of* G*.*

Proof. Firstly, each of Lines 1–4 runs in $\mathcal{O}(m)$ time. Specifically, VertexReduction, degree-based pruning and degeneracy ordering can all be computed in $\mathcal{O}(m)$ time. The time complexity of MBC-Heu (i.e., Line 2) is also $\mathcal{O}(m)$, which will be discussed in Section 5.3.3. Secondly, the for loop at Line 5 processes each vertex u, by constructing the dichromatic network g_u of u and then computing the maximum dichromatic clique in g_u. The maximum dichromatic clique in g_u is computed by invoking MDC, which then recursively invokes itself (Line 21). Note that, the number of vertices in g_u is bounded by δ (Chang, 2019). Thus, for a fixed dichromatic network g_u, the total number of invocations to MDC is at most 2^{δ}, since each invocation to MDC has a different input C. It is easy to see that each invocation to MDC (excluding Line 21) takes $\mathcal{O}(|E(g)|) = \mathcal{O}(m)$ time; note that, the coloring-based upper bound can be computed in linear time to the number of edges. Therefore, the time complexity of Algorithm 17 is $\mathcal{O}(n \cdot m \cdot 2^{\delta})$.

For the space complexity, the input graph G takes $\mathcal{O}(m)$ space and the dichromatic network g_u takes $\mathcal{O}(m)$ space. Note that, although MDC takes a graph g as input, we do not store a separate copy of g for each invocation to MDC. Instead, we only store g_u in the main memory and modify g_u before going to the recursion of Line 21, and after returning from the recursion, we restore the graph g_u, in the same way as (Eppstein et al., 2013). Moreover, after processing a dichromatic network, we

release its memory, and thus there will be at most one dichromatic network stored in the main memory through out the execution of Algorithm 17. Thus, the space complexity is $\mathcal{O}(m)$. □

5.3.3 A Heuristic Algorithm MBC-Heu

In this subsection, we present a heuristic algorithm MBC-Heu for the maximum balanced clique problem, which is invoked at Line 2 of Algorithm 17 for initialization. Let u be the vertex of maximum degree in the input graph G. We extract the dichromatic network g_u of u, and then iteratively grow a clique C in g_u. That is, C is initialized as $\{u\}$. Let g be the subgraph of g_u induced by the set of vertices that are adjacent to all vertices of C. We greedily include into C the vertex that has the maximum degree in g, and then update g; we repeat this process until g is empty. To balance the sizes of $|C \cap V_L|$ and $|C \cap V_R|$, we add vertices into C from V_L and from V_R in an alternating order.

Algorithm 18: MBC-Heu

Input: A signed graph $G = (V, E^+, E^-)$, and a threshold τ
Output: A balanced clique C satisfying the constraint τ or $\emptyset$

```
1  u ← maximum-degree vertex in G;
2  g ← the dichromatic network g_u of u;
3  C ← {u};
4  while V_L(g) ≠ ∅ or V_R(g) ≠ ∅ do
5      if V_L(g) = ∅ or (V_R(g) ≠ ∅ and |C_L| ≥ |C_R|) then
6          v ← maximum-degree vertex of V_R(g) in g;
7      else v ← maximum-degree vertex of V_L(g) in g;
8      C ← C ∪ {v}; g ← g[N_g(v)];
9  if |C_L| ≥ τ and |C_R| ≥ τ then return C;
10 else return ∅;
```

The pseudocode of our heuristic algorithm MBC-Heu is shown in Algorithm 18, which is self-explanatory. Note that, we return the clique C only when it satisfies the polarization constraint τ (Lines 9–10). In our implementation, we run the heuristic algorithm for the vertex u with the largest value of $\min\{d^+(u), d^-(u)\}$.

Theorem 8. *The time complexity and space complexity of* MBC-Heu *(Algorithm 18) are both* $\mathcal{O}(m)$.

Proof. Line 1 takes $\mathcal{O}(m)$ time to obtain the maximum-degree vertex u. Line 2 takes $\mathcal{O}(m)$ time to construct the dichromatic network g_u. The while loop is the most time-critical. To efficiently obtain the maximum-degree vertex v in the while loop (i.e., Lines 6,7), we maintain the degree in g for each vertex of g. Without loss of generality, assume $v_1, \ldots, v_l$ are the maximum-degree vertices that are iteratively obtained in the while loop. It is easy to see that the total time complexity of Lines 6,7 for the entire execution of while loop is $\mathcal{O}(n + \sum_{i=1}^{l} d_G(v_i)) = \mathcal{O}(m)$; note that the number of vertices in g is at most $d_G(v_i)$ after v_i is added to C, see Line 8. Also, the total time complexity of maintaining the degree information for vertices of

g is $\mathcal{O}(m)$, as each vertex is removed from g at most once at Line 8. As a result, the time complexity of Algorithm 18 is $\mathcal{O}(m)$. For the space complexity, it is immediate that it is $\mathcal{O}(m)$, as we iteratively modify g; that is, there is only one copy of g during the execution. □

5.4 Large Balanced Clique Enumeration

In this section, we study the large balanced clique enumeration problem, which aims to enumerate all maximal balanced cliques C satisfying the constraint τ such that $|C| \geq \omega_\tau(G) - \alpha$ for user-given thresholds τ and α.

Problem 5 (Large Balanced Clique Enumeration)**.** Given a signed graph G and two non-negative thresholds τ and α, the large balanced clique enumeration problem aims to enumerate all maximal balanced cliques $C \subseteq V$ satisfying the constraint τ such that $|C| \geq \omega_\tau(G) - \alpha$. A balanced clique is maximal if none of its proper supersets is balanced.

Theorem 9. *The large balanced clique enumeration problem is #P-complete.*

Proof. This theorem directly follows from the facts that (1) the problem of enumerating all maximal balanced cliques is #P-complete (Chen et al., 2020), and (2) it is a special case of our problem, i.e., by setting α to be a very large value (e.g., $\alpha = |V|$). □

5.4.1 A Baseline Approach BCE

Recall that the MBCEnum algorithm proposed in (Chen et al., 2020) enumerates all maximal balanced cliques satisfying the constraint τ. A baseline algorithm can be designed by adapting MBCEnum to enumerate all large balanced cliques. A straightforward adaptation would be first invoking MBCEnum to enumerate all maximal balanced cliques and then reporting those of size at least $|C^*| - \alpha$, where C^* is the largest one among all identified maximal balanced cliques. It is easy to observe that much computation is wasted on detecting small maximal balanced cliques in this straightforward adaptation.

To improve the efficiency, we incorporate pruning techniques into the enumeration process such that small maximal balanced cliques are not enumerated. That is, let C^* be the largest balanced clique found so far, then we use $|C^*| - \alpha$ as a lower bound to prune search branches that will not generate large balanced cliques; note that C^* keeps updating during the enumeration process and is a maximum balanced clique when the algorithm terminates. Recall from Section 5.3.1 that MBCEnum grows a balanced clique $C_L \cup C_R$ by iteratively adding candidate vertices from P_L to C_L and adding candidate vertices from P_R to C_R. Thus, in a search branch with inputs (C_L, C_R, P_L, P_R), the size of the largest balanced clique that can be found through recursions of this search branch is at most $|C_L|+|C_R|+|P_L|+|P_R|$. Consequently, we can safely terminate this search branch if its upper bound $|C_L| + |C_R| + |P_L| + |P_R|$ is smaller than $|C^*| - \alpha$. We denote this baseline algorithm as BCE.

5.4.2 A Dichromatic Clique-based Approach BCE*

Due to the same drawbacks as MBC for computing a maximum balanced clique, BCE is still inefficient for enumerating all large balanced cliques. In this subsection, we propose a dichromatic clique-based approach BCE* for efficiently enumerating all large balanced cliques, by following the general idea of MBC* that is introduced in Section 5.3.2.

Following the arguments in Section 5.3.2, we know that for any balanced clique C containing u, it must satisfy $C_L \subseteq \{u\} \cup N_G^+(u)$ and $C_R \subseteq N_G^-(u)$; here, without loss of generality, we assume $u \in C_L$. Therefore, all large balanced cliques containing u can be found in the subgraph G_u of G induced by vertices $\{u\} \cup N_G^+(u) \cup N_G^-(u) = \{u\} \cup N_G(u)$. In addition, all conflicting edges can be removed from G_u such that we can ignore edge signs and transform G_u into a dichromatic-network g_u, in the same way as in Section 5.3.2. Let $V_L = \{u\} \cup N_G^+(u)$ be the L-vertices and $V_R = N_G^-(u)$ be the R-vertices. It is easy to verify that *each balanced clique C containing u in G is a dichromatic clique containing u in g_u where $C_L = C \cap V_L$ and $C_R = C \cap V_R$, and vice versa*. Consequently, our problem becomes enumerating large maximal dichromatic cliques in g_u; note that every maximal clique in g_u must contain u, due to the way of constructing g_u. We term this problem as large dichromatic clique enumeration problem, formally defined as follows.

Problem 6 (Large Dichromatic Clique Enumeration)**.** Given a *dichromatic graph* $g = (V(g), E(g))$ where $V(g) = V_L \cup V_R$ with $V_L \cap V_R = \emptyset$, and non-negative thresholds τ and λ, the large dichromatic clique enumeration problem aims to enumerate all maximal cliques C in g such that $|C| \geq \lambda$, $|C \cap V_L| \geq \tau$, and $|C \cap V_R| \geq \tau$.

To obtain all large balanced cliques in G, we can enumerate all large dichromatic cliques in the dichromatic-network g_u for each $u \in V(G)$. However, an obvious issue is that a maximal dichromatic clique C will be reported multiple times. That is, C will be reported when processing the dichromatic-network g_u for each $u \in C$. This also means that a lot of computations are wasted. To avoid duplication and to speed up the computation, we define a total ordering $\prec$ for all vertices of G — without loss of generality, assume the ordering is $\{v_1, v_2, \ldots, v_n\}$ — and revise the definition of dichromatic-network g_{v_i} to be the subgraph of G induced by v_i and v_i's higher-ranked neighbors, i.e., induced by $\{v_i\} \cup (N_G(v_i) \cap \{v_{i+1}, \ldots, v_n\})$. Still, conflicting edges are removed from g_{v_i} and its vertices are labeled as L-vertices and R-vertices. Then, when processing g_{v_i}, we only aim to enumerate all maximal balanced cliques C of G such that $v_i \in C$ and v_i ranks the lowest among all vertices of C according to $\prec$. Note however that, due to the revised definition of g_{v_i}, a maximal clique C in g_{v_i} may not be maximal in G; specifically, some of v_i's lower-ranked neighbors, i.e., $N_G(v_i) \cap \{v_1, v_2, \ldots, v_{i-1}\}$, may be able to be added to enlarge C. To check whether a maximal clique C in g_{v_i} is also maximal in G, we maintain the set of common neighbors of C in G, denoted Λ_C. To be precise, Λ_C is the set of vertices of G such that each vertex $x \in \Lambda_C$ (i) either has a positive edge to every vertex of $C \cap V_L$ and a negative edge to every vertex of $C \cap V_R$, (ii) or has a positive edge to every vertex of $C \cap V_R$ and a negative edge to every vertex of $C \cap V_L$. As a result, C is maximal if and only if $\Lambda_C = \emptyset$.

Based on the above discussions, we transform the large balanced clique enumeration problem over a signed graph to a series of large dichromatic clique enumeration problems over unsigned dichromatic-networks. The pseudocode of our algorithm for enumerating all large balanced cliques is given in Algorithm 19, denoted BCE*. We first compute the maximum balanced clique C^* in G by invoking MBC* (Line 1), and derive the size threshold of large balanced cliques as the larger one between $|C^*| - \alpha$ and 2τ, denoted as λ (Line 2); note that, a balanced clique satisfying the constraint τ must be of size at least 2τ. Then, we prune unqualified vertices from G based on λ and τ (Line 3). That is, any vertex in a large balanced clique must have at least $\tau - 1$ positive neighbors, at least τ negative neighbors and at least $\lambda - 1$ neighbors in total; thus, any vertex violating either of these three conditions can be safely removed from G. Next, we compute a degeneracy ordering for vertices of G (Line 4) — without loss of generality, we assume the degeneracy ordering is $\{v_1, v_2, \ldots, v_n\}$ — and process vertices in this order (Line 5). When processing vertex v_i, we construct the dichromatic-network g_{v_i} (Line 6), which is the subgraph of G induced by $\{v_i\} \cup (N_G(v_i) \cap \{v_{i+1}, \ldots, v_n\})$ after removing conflicting edges and discarding edge signs, and enumerate all maximal balanced cliques of G that have v_i as the lowest-ranked vertex, by invoking the procedure DCE (Line 9). Before invoking DCE, we first reduce the size of g_{v_i} based on λ and τ (Line 7), and compute a coloring-based upper bound for the maximum clique size in g_{v_i} (Line 8); if the upper bound is smaller than λ, then g_{v_i} will contain no large balanced cliques. The reduction at Line 7 is similar to that at Line 3, with subtle differences. Specifically, a vertex in V_L (resp. V_R) can be removed from g_{v_i} if it has either less than $\tau - 1$ neighbors in V_L (resp. V_R), or less than τ neighbors in V_R (resp. V_L), or less than $\lambda - 1$ total neighbors in $V_L \cup V_R$.

The pseudocode of DCE is also given in Algorithm 19. The input of DCE consists of vertex subsets $C \subseteq V(g_{v_i})$ and $X \subseteq V$, a subgraph g, and three thresholds τ_L, τ_R, λ. The algorithm maintains the invariants that (1) C, $V(g)$, and X are disjoint, (2) $V(g) \cup X$ is the set of common neighbors of C, i.e., $V(g) \cup X = \Lambda_C$ as defined above, such that C is maximal if and only if $V(g) \cup X = \emptyset$, and (3) $C \cup C'$ for $C' \subseteq V(g)$ is a large maximal dichromatic clique in g_{v_i} if and only if C' is a maximal dichromatic clique in g such that $|C' \cap V_L(g)| \geq \tau_L$, $|C' \cap V_R(g)| \geq \tau_R$, $|C'| \geq \lambda$. DCE works as follows. Firstly, C is a large maximal balanced clique in G if $V(g) \cup X = \emptyset$, and $\tau_L \leq 0, \tau_R \leq 0$ and $\lambda \leq 0$ (Lines 10–11). If C is not maximal, then we reduce g based on τ_L, τ_R, λ and conduct upper bound-based pruning by using the coloring-based upper bound (Lines 12–13), in a similar way to Lines 7–8. After that, we pick a pivot vertex from $V(g) \cup X$, which is the vertex that has the largest number of neighbors in g (Line 14). Note that g is a subgraph of g_{v_i}. For a vertex that is in g_{v_i}, its set of neighbors in g is trivially defined based on g_{v_i}. For a vertex v that is not in g_{v_i}, i.e., $v \in N_G(v_i) \cap \{v_1, \ldots, v_{i-1}\}$ that is passed to DCE at Line 9, its neighbors in g_{v_i} and thus in g are defined as follows where conflicting edges between $N_G(v_i) \cap \{v_1, \ldots, v_{i-1}\}$ and $V(g_{v_i})$ are removed: if v is a positive neighbor v_i in G, then v's neighbors in g_{v_i} are defined as the positive neighbors in $V_L(g_{v_i})$ and negative neighbors in $V_R(g_{v_i})$; otherwise, v's neighbors in g_{v_i} are defined as the positive neighbors in $V_R(g_{v_i})$ and negative neighbors in $V_L(g_{v_i})$. Let $v^* \in V(g) \cup X$ be the pivot vertex. Then, it is easy to see that any maximal clique in g must contain a

Algorithm 19: BCE*

Input: A signed graph G, and thresholds τ and α
Output: All large maximal balanced cliques in G

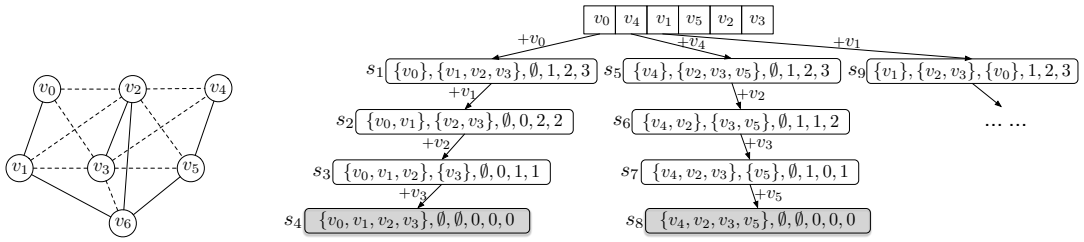
1 $C^* \leftarrow \mathsf{MBC}^*(G, \tau)$;
2 $\lambda \leftarrow \max\{|C^*| - \alpha, 2\tau\}$;
3 Reduce G based on λ and τ;
4 Compute a degeneracy ordering for vertices of G; w.l.o.g., assume the ordering is $\{v_1, v_2, \ldots, v_n\}$;
5 **for each** *vertex* $v_i \in \{v_1, v_2, \ldots, v_n\}$ **do**
6 $\quad g_{v_i} \leftarrow$ the dichromatic-network of v_i induced by $\{v_i\} \cup (N_G(v_i) \cap \{v_{i+1}, \ldots, v_n\})$;
7 $\quad$ Reduce g_{v_i} based on λ and τ;
8 $\quad$ **if** $v_i \notin V(g_{v_i})$ **or** $\mathsf{colorUB}(g_{v_i}) < \lambda$ **then continue**;
9 $\quad \mathsf{DCE}(\{v_i\}, g_{v_i} \backslash \{v_i\}, N_G(v_i) \cap \{v_1, \ldots, v_{i-1}\}, \tau - 1, \tau, \lambda - 1)$;

Procedure $\mathsf{DCE}(C, g = (V_L, V_R, E), X, \tau_L, \tau_R, \lambda)$
10 **if** $g = \emptyset$ **and** $X = \emptyset$ **and** $\tau_L \leq 0$ **and** $\tau_R \leq 0$ **and** $\lambda \leq 0$ **then**
11 $\quad$ **report** C as a large maximal balanced clique;
12 Reduce g based on λ, τ_L and τ_R;
13 **if** $\mathsf{colorUB}(g) < \lambda$ **then return**;
14 $v^* \leftarrow \arg\max_{v \in X \cup V(g)} |N_g(v)|$;
15 **for each** *vertex* $v \in V(g) \setminus N_g(v^*)$ **do**
16 $\quad$ **if** $v \in V_L(g)$ **then** $\tau'_L \leftarrow \tau_L - 1$; $\tau'_R \leftarrow \tau_R$;
17 $\quad$ **else** $\tau'_L \leftarrow \tau_L$; $\tau'_R \leftarrow \tau_R - 1$;
18 $\quad \mathsf{DCE}(C \cup \{v\}, g[N_g(v)], X \cap N(v), \tau'_L, \tau'_R, \lambda - 1)$;
19 $\quad g \leftarrow g \backslash \{v\}$; $X \leftarrow X \cup \{v\}$;

non-neighbor of v^*. Thus, we only need to generate new branches for each vertex of $V(g) \setminus N_g(v^*)$ (Line 15). Note that, although DCE shares similar spirit to the pivoted version of the BronKerbosch algorithm (Bron and Kerbosch, 1973) that enumerates all maximal cliques in an unsigned graph, DCE is different by considering dichromatic cliques and incorporating pruning techniques at Lines 12–13. Moreover, our idea of transforming the large balanced clique enumeration problem to large dichromatic clique enumeration problems that do not need to consider edge signs is new.

Example 17. *A snippet of the search tree of running* BCE* *on the signed graph in Figure 5.5(a) with* $\tau = 2$ *and* $\alpha = 0$ *is shown in Figure 5.5(b), where each rectangle is a search state* s_i *represented by* $(C, V(g), X, \tau_L, \tau_R, \lambda)$. *Firstly, we invoke* MBC* *to compute the maximum balanced clique* $C^* = \{v_0, v_1, v_2, v_3\}$, *and derive the size threshold* λ *as* $\max\{|C^*| - \alpha, 2\tau\} = 4$. *Then we reduce the graph based on* λ *and* τ; *in this step,* v_6 *will be pruned since its negative degree is less than* 2. *Next, the degeneracy ordering for the remaining vertices is computed as* $\{v_0, v_4, v_1, v_5, v_2, v_3\}$, *and we process them in this order.*

Now, let us see the enumeration process for v_0. *We construct the dichromatic-network* g_{v_0}, *which is the subgraph of* G *induced by* $\{v_0, v_1, v_2, v_3\}$ *by discarding the conflicting edges and edge signs. Then, we start to find large dichromatic cliques in* g_{v_0} *by invoking* DCE. *As shown in state* s_1, C *is initialized as* $\{v_0\}$ *and* $V(g)$ *is*



(a) A signed graph (b) A snippet of the search tree of BCE* ($\tau = 2$ and $\alpha = 1$)

Figure 5.5: Illustration of BCE*

initialized as $\{v_1, v_2, v_3\}$. *Note that* v_1 *is an* L*-vertex since it is a positive neighbor of* v_0, v_2 *and* v_3 *are* R*-vertices since they are negative neighbors of* v_0. X *is empty since* v_0 *is the lowest-ranked vertex.* τ_L, τ_R *and* λ *are* 1, 2 *and* 3, *respectively. For* s_1, *vertex* v_1 *is chosen as the pivot. Since both* v_2 *and* v_3 *are adjacent to the pivot* v_1, *only one new branching is generated for* s_1, *i.e.,* v_1 *is added into* C *and we reach state* s_2. *After adding* v_1 *to* C, *both* τ_L *and* λ *are decreased by* 1 *since* v_1 *is an* L*-vertex. For* s_2, *vertex* v_2 *is selected as the pivot and we only generate one new branching by adding* v_2 *into* C *(i.e., state* s_3*). Similarly, we continue the expansion by adding* v_3 *into* C *and reach the maximal balanced clique* $C = \{v_0, v_1, v_2, v_3\}$ *at state* s_4. *In a similar way, we enumerate large balanced cliques for other vertices. Finally, we find two large balanced cliques in this graph:* $C_1 = \{v_0, v_1, v_2, v_3\}$ *(at state* s_4*) and* $C_2 = \{v_2, v_3, v_4, v_5\}$ *(at state* s_8*).*

Theorem 10. *The time complexity of* BCE* *(Algorithm 19) excluding Line 1 is* $\mathcal{O}(\delta \cdot n \cdot 3^{\delta/3})$ *and the space complexity is* $\mathcal{O}(m)$, *where* δ *is the degeneracy of* G.

Proof. Line 2 runs in constant time, and each of Line 3 and Line 4 runs in $\mathcal{O}(m)$ time. Then, we enumerate all large balanced cliques for each vertex v_i by invoking DCE (Lines 5–9). As each dichromatic-network g_{v_i} contains at most δ vertices (i.e., $|N_G(v_i) \cap \{v_{i+1}, \ldots, v_n\}| \leq \delta$), Lines 6–8 run in $\mathcal{O}(\delta^2)$ time. Also note that Lines 11–13 run in $\mathcal{O}(|V(g)|^2)$ time. Thus, the total time complexity of Algorithm 19 excluding Line 1 is $\mathcal{O}(\delta \cdot n \cdot 3^{\delta/3})$ by following the arguements of (Eppstein et al., 2013). In addition, the space complexity of Algorithm 19 is $\mathcal{O}(m)$ which can be shown in the same way as the proof of Theorem 7. □

5.5 Polarization Factor Computation

In this section, we study the polarization factor computation problem.

Problem 7 (Polarization Factor Problem)**.** Given a signed graph $G = (V, E^+, E^-)$, the polarization factor problem aims to compute the largest τ^* such that G has a balanced clique satisfying the polarization constraint τ^*.

We call this largest τ^* the *polarization factor* of G, and denote it as $\beta(G)$. For example, the polarization factor of the signed graph in Figure 5.2 is 3.

Based on $\beta(G)$, we also extend our techniques to solve the maximum balanced clique problem for every $0 \leq \tau \leq \beta(G)$, and thus do not require end-users to input a threshold τ. Note that, there will be no result for $\tau > \beta(G)$.

Theorem 11. *The polarization factor computation problem is NP-hard.*

Proof. We can prove the NP-hardness by reducing from the classic maximum clique problem over unsigned graphs which is NP-hard (Karp, 1972), which is same as the proof of the maximum balanced clique computation problem (i.e., Theorem 5). Thus, we omit the details. □

5.5.1 An Enumeration-based Baseline Algorithm PF-E

Recall that the polarization factor of a signed graph $G = (V, E^+, E^-)$, denoted $\beta(G)$, is the largest τ such that G has a balanced clique C satisfying $|C_L| \geq \tau$ and $|C_R| \geq \tau$. Similar to Section 5.3.1, we can modify the maximal balanced clique enumeration algorithm MBCEnum (Chen et al., 2020) to compute $\beta(G)$. We denote this enumeration-based algorithm as PF-E. The correctness of PF-E is immediate. However, the time cost of PF-E is expensive as one would expect.

5.5.2 A Binary Search Algorithm PF-BS

It is obvious that for any $\tau \leq \beta(G)$, we can always find a balanced clique in G satisfying the constraint τ, and for any $\tau > \beta(G)$, we will not be able to find any. Thus, we can iteratively invoke our algorithm MBC* proposed in Section 5.3.2 for increasing τ to compute $\beta(G)$. That is, we stop when MBC* is not able to find any balanced clique satisfying the constraint τ, and then $\tau - 1$ is the result.

A better approach is binary searching the value of $\beta(G)$, which also invokes MBC* for each guessing τ of $\beta(G)$. The lower bound of $\beta(G)$ is 0, and the upper bound can be set as $\max_{v \in V}\{\min\{d^+(v) + 1, d^-(v)\}\}$. We denote our binary search-based algorithm as PF-BS, and we omit the pseudocode. The correctness of PF-BS is immediate.

Optimization. MBC* was designed to find the maximum balanced clique satisfying the polarization constraint τ. However, for the polarization factor problem, we only need to know whether there exists a balanced clique satisfying the polarization constraint τ or not, i.e., the balanced clique does not need to be *maximum*. Thus, in each invocation of MBC*, we can terminate the execution of MBC* once both τ_L and τ_R (see Algorithm 17) decrease to 0.

5.5.3 A Dichromatic Clique Checking-based Algorithm PF*

The PF-BS algorithm invokes MBC* as a black box, and needs to invoke MBC* $\mathcal{O}(\log n)$ times in the worst case. In this subsection, we look into the algorithm MBC*, and directly adapt MBC* for computing the polarization factor $\beta(G)$.

Firstly, similar to Theorem 6, we have the lemma below that transforms the polarization factor problem over G to a series of another problem over small subgraphs of G. Here, for each dichromatic network g_u, we compute the largest τ such that

g_u has a dichromatic clique satisfying the constraint τ, instead of computing the maximum dichromatic clique for a given τ.

Lemma 15. *For any signed graph $G = (V, E^+, E^-)$ and a total ordering $\prec$ of V, we have $\beta(G) = \max_{u \in V} \gamma(g_u)$, where $g_u = (V_L, V_R, E)$ is the dichromatic network of u as defined in Section 5.3.2, and $\gamma(g_u)$ denotes the largest τ such that g_u has a dichromatic clique satisfying the constraint τ.*

Proof. This lemma can be proved in a similar way to the proof of Theorem 6. We omit the details. □

Let $(v_1, v_2, \ldots, v_n)$ be any total ordering of V. We process vertices in the reverse order according to the total ordering. Following Lemma 15, when processing vertex v_i, we need to compute the largest τ such that g_{v_i} has a dichromatic clique satisfying the constraint τ. Actually, we can do better. The general idea is that let $\tau = \max_{j=i+1}^{n} \gamma(g_{v_j})$, then we only need to verify whether g_{v_i} has a dichromatic clique satisfying the constraint $\tau + 1$; we call this problem the *dichromatic clique checking* problem. This is because $\gamma(g_{v_i})$ cannot be larger than $\tau + 1$, as proved in the lemma below.

Lemma 16. *Let $(v_1, v_2, \ldots, v_n)$ be a total ordering of V. We have $\gamma(g_{v_i}) \leq \max_{j=i+1}^{n} \gamma(g_{v_j}) + 1$ for all $i \geq 1$.*

Proof. Let C be a dichromatic clique in $g_{v_i} = (V_L, V_R, E)$ satisfying $|C \cap V_L(g_{v_i})| \geq \gamma(g_{v_i})$ and $|C \cap V_R(g_{v_i})| \geq \gamma(g_{v_i})$. Let $C' = C \setminus \{u\}$, and suppose v_j is the lowest-ranked vertex in C'. It is easy to verify that $j > i$ and C' is a dichromatic clique in g_{v_j} satisfying $|C' \cap V_L(g_{v_j})| \geq \gamma(g_{v_i}) - 1$ and $|C' \cap V_R(g_{v_j})| \geq \gamma(g_{v_i}) - 1$. Thus, the lemma holds. □

The pseudocode of our algorithm is given in Algorithm 20, denoted PF*. First, we heuristically compute a polarization value by invoking MBC-Heu$(G, 0)$ (Line 1). Note that, MBC-Heu$(G, 0)$ actually returns a balanced clique C; we set $\tau^* = \min\{|C_L|, |C_R|\}$. Based on τ^*, we reduce the input graph G by applying VertexReduction of (Chen et al., 2020) (Line 2). Next, we compute a total ordering POrder of V by invoking PDecompose (Line 3). Based on the total ordering POrder, we process vertices in reverse order according to POrder (Line 4). For each vertex u, we extract the dichromatic-network g_u of u (Line 5), and then reduce g_u to its $(\tau^* + 1, \tau^* + 1)$-core (Line 6), denoted g. If u is not pruned (Line 7), then we invoke DCC to check whether g has a dichromatic clique C satisfying $|C \cap V_L(g)| \geq \tau^* + 1$ and $|C \cap V_R(g)| \geq \tau^* + 1$; if the answer is yes, then we increase τ^* by 1 (Line 8). Here, the (τ_L, τ_R)-core for non-negative integers τ_L and τ_R is the (unique) maximal subgraph $g = (V_L, V_R, E)$ such that every vertex of V_L (resp. V_R) has at least $\tau_L - 1$ (resp. τ_L) neighbors in V_L and at least τ_R (resp. $\tau_R - 1$) neighbors in V_R. The motivation of computing the (τ_L, τ_R)-core is that every vertex participating in a dichromatic clique C satisfying $|C \cap V_L(g_u)| \geq \tau_L$ and $|C \cap V_R(g_u)| \geq \tau_R$ must be in the (τ_L, τ_R)-core. Note that, the (τ_L, τ_R)-core can be computed in linear time by iteratively removing vertices that violate the conditions.

The pseudocode of DCC is also given in Algorithm 20. It is similar to MDC in Algorithm 17, except that (1) we now do not need to keep track of the growing

Algorithm 20: PF*

Input: A signed graph $G = (V, E^+, E^-)$
Output: The polarization factor $\beta(G)$

1 $\tau^* \leftarrow$ MBC-Heu$(G, 0)$;
2 Reduce G by VertexReduction of (Chen et al., 2020) based on $\tau^* + 1$;
3 POrder$(\cdot) \leftarrow$ PDecompose(G);
4 **for each** *vertex u in reverse order w.r.t.* POrder$(\cdot)$ **do**
5 $\quad g_u \leftarrow$ the dichromatic-network of u;
6 $\quad g \leftarrow$ the $(\tau^* + 1, \tau^* + 1)$-core of g_u;
7 $\quad$ **if** $u \in V_L(g)$ **then**
8 $\quad\quad$ **if** DCC$(g \setminus \{u\}, \tau^*, \tau^* + 1)$ **then** $\tau^* \leftarrow \tau^* + 1$;
9 **return** τ^*;

Procedure DCC$(g = (V_L, V_R, E), \tau_L, \tau_R)$
Output: **true** if g contains a valid dichromatic clique for thresholds τ_L and τ_R; **false** otherwise

10 **if** $\tau_L = 0$ **and** $\tau_R = 0$ **then return true**;
11 Reduce g to its (τ_L, τ_R)-core;
12 **if** $\tau_L > 0$ **and** $\tau_R = 0$ **then** $\mathcal{B} \leftarrow V_L(g)$;
13 **else if** $\tau_L = 0$ **and** $\tau_R > 0$ **then** $\mathcal{B} \leftarrow V_R(g)$;
14 **else** $\mathcal{B} \leftarrow V_L(g) \cup V_R(g)$;
15 **while** $\mathcal{B} \neq \emptyset$ **do**
16 $\quad v \leftarrow$ the vertex of $\mathcal{B}$ with the minimum degree in g;
17 $\quad$ **if** $v \in V_L(g)$ **then** $\tau'_L \leftarrow \tau_L - 1$; $\tau'_R \leftarrow \tau_R$;
18 $\quad$ **else** $\tau'_L \leftarrow \tau_L$; $\tau'_R \leftarrow \tau_R - 1$;
19 $\quad$ **if** DCC$(g[N_g(v)], \tau'_L, \tau'_R)$ **then return true**;
20 $\quad$ Remove v from $\mathcal{B}$ and g;
21 **return false**;

clique C, and (2) we can stop the execution (Lines 10 and 20) once both τ_L and τ_R become 0; note that, in DCC, τ_L and τ_R will not go below 0, while it is possible in MDC. It is worth pointing out that we first greedily add u to the clique C at Line 8 before invoking DCC; this is because if u is not in the clique, then invoking DCC definitely will return false according to Lemma 16.

Theorem 12. *The time complexity of* PF* *(Algorithm 20) is* $\mathcal{O}(n \cdot m \cdot 2^\delta)$ *if we use the degeneracy ordering at Line 3. The space complexity is* $\mathcal{O}(m)$.

Proof. The time and space complexity of Algorithm 20 follows in the same way as that of Algorithm 17, if we use the degeneracy ordering at Line 3 of Algorithm 20. Note that, although we do not explicitly pass the growing clique C as the input to DCC, such a growing clique C is implicitly maintained (i.e., by adding v obtained at Line 16 of Algorithm 20). Thus, the total number of invocations to DCC for each fixed dichromatic network g_u is bounded by 2^δ. □

Polarization Order. Any total ordering of V can be used at Line 4 of Algorithm 20 for correctly computing $\beta(G)$. One possibility is to use the degeneracy ordering as

used in Algorithm 17. In the following, we propose a new total ordering, called polarization order, to improve the time efficiency. First, we define the polarization core.

Definition 11 (Polarization Core)**.** Given a signed graph $G = (V, E^+, E^-)$ and an integer k, the k-polarization core (abbreviated as k-polar-core) of G is the maximal subgraph g of G such that $\min\{d_g^+(u) + 1, d_g^-(u)\} \geq k$, $\forall u \in V(g)$.

We define the *polar-core number* of a vertex u, denoted $pn(u)$, as the largest k such that u is contained in the k-polar-core. The reason of defining these concepts is that $pn(u)$ provides an upper bound of $\gamma(g_u)$ for any total ordering $\prec$, as provided in the lemma below. Consequently, if we process vertices in non-increasing polar-core number, we are likely to obtain a large lower bound of $\beta(G)$ by processing the first few ego-networks (or dichromatic networks), and thus prune a lot of invocations to DCC based on the lower bound of $\beta(G)$.

Lemma 17. *For any vertex u in G, we have $pn(u) \geq \gamma(g_u)$ for any total ordering $\prec$.*

Proof. Recall that g_u is the dichromatic-network of u as defined in Section 5.3.2, and $\gamma(g_u)$ is the largest τ such that g_u has a dichromatic clique satisfying the constraint τ. Let C be a dichromatic clique in g_u satisfying the constraint $\gamma(g_u)$. Then, for every vertex $v \in C$, we have $\min\{d_C^+(v) + 1, g_C^-(v)\} \geq \gamma(g_u)$; moreover, this also holds for $\{u\} \cup C$. Consequently, $pn(u) \geq \gamma(g_u)$, and the lemma holds. □

Algorithm 21: PDecompose

Input: A signed graph $G = (V, E^+, E^-)$
Output: Polarization order POrder

```
for each v ∈ V do
    d⁺(v) ← the positive degree of v in G;
    d⁻(v) ← the negative degree of v in G;
POrder ← ∅;
for i ← 1 to n do
    u ← arg min_{v∈V\POrder} min{d⁺(v) + 1, d⁻(v)};
    pn(u) ← min{d⁺(u) + 1, d⁻(u)};
    Add u to the tail of POrder;
    for each positive neighbor v of u that not in POrder do
        if d⁺(v) + 1 > pn(u) then d⁺(v) ← d⁺(v) − 1;
    for each negative neighbor v of u that not in POrder do
        if d⁻(v) > pn(u) then d⁻(v) ← d⁻(v) − 1;
return POrder;
```

We call the total ordering that ranks vertices in non-decreasing polar-core number the *polarization order*. The polarization order can be computed in a similar way to the peeling algorithm for computing the degeneracy ordering (Matula and Beck, 1983). The pseudocode code of our algorithm, denoted PDecompose, is shown in

Algorithm 21, which is self-explanatory. The general idea is to iteratively remove from G the vertex u that has the smallest $\min\{d^+(u)+1, d^-(u)\}$ in the current graph, and add u to the end of POrder. By using the bin-sort like data structure as discussed in (Matula and Beck, 1983), PDecompose can be implemented to run in $\mathcal{O}(n+m)$ time and $\mathcal{O}(m)$ space; we omit the details.

5.6 Maximum Balanced Clique for Every τ

In this section, we investigate the problem of computing a maximum balanced clique for each $\tau \geq 0$, which then eliminates the need for end-users to specify the threshold τ. We term this problem as the *generalized maximum balanced clique problem.*

A straightforward method is to invoke MBC* independently for each τ from 0 to $\beta(G)+1$. Note that, as $\beta(G)$ is not an input to this problem, we need to invoke MBC* for $\tau = \beta(G)+1$ which would return an empty result and thus indicate that this τ is larger than $\beta(G)$. We denote this algorithm as gMBC. We can do better by sharing computation among the different invocations to MBC*, based on the lemma below.

Lemma 18. *Given a signed graph G and thresholds τ_1 and τ_2 s.t. $0 \leq \tau_1 < \tau_2 \leq \beta(G)$, the maximum balanced clique for threshold τ_1 is no smaller than that for τ_2.*

Proof. Let C^{τ_2} be the maximum balanced clique for threshold τ_2. It is obvious that C^{τ_2} also satisfies the threshold τ_1 since $\tau_1 < \tau_2$. Consequently, the maximum balanced clique for threshold τ_1 will be no smaller than C^{τ_2}. □

Algorithm 22: gMBC*

Input: A signed graph $G = (V, E^+, E^-)$
Output: A maximum balance clique for each $0 \leq \tau \leq \beta(G)$

```
1 Set τ as β(G) by invoking PF*;
2 while τ ≥ 0 do
3     if τ = β(G) then g ← reduce G to (2τ − 1)-core;
4     else g ← reduce G to |C^{τ+1}|-core;
5     C^τ ← MBC*(g, τ);
6     if C^τ = ∅ then C^τ ← C^{τ+1};
7     τ ← τ − 1;
8 return {C^0, ..., C^{β(G)}};
```

Based on the result in Lemma 18, we propose to compute maximum balanced cliques for decreasing τ value order. Suppose we have obtained the maximum balanced clique C^τ. Then we can use C^τ as the initial solution to problem for threshold $\tau-1$. In this way, the search space of computing the maximum balanced clique for $\tau-1$ will be greatly reduced, since we observe that in practice $|C^\tau| = |C^{\tau-1}|$ for many of the τ values. Note that, however, we need to first invoke PF* to get the polarization factor $\beta(G)$ and set the initial τ as $\beta(G)$. We denote this approach as gMBC*. The pseudocode of gMBC* is shown in Algorithm 22. It is immediate that the time complexity of gMBC* is $\mathcal{O}(\beta(G) \cdot n \cdot m \cdot 2^\delta)$ and the space complexity is $\mathcal{O}(m)$.

5.7 Experimental Results

In this section, we empirically evaluate the efficiency and effectiveness of our algorithms. For the maximum balanced clique computation problem, we implemented MBC (Algorithm 16), MBC* (Algorithm 17), as well as three variants of these two algorithms: (1) MBC-noER which is MBC without EdgeReduction, (2) MBC-Adv which is the improved version of MBC that applies the degree-based pruning and coloring-based upper bound by ignoring edge signs, and (3) MBC*-withER which is the variant of MBC* that also applies EdgeReduction at Line 1 of Algorithm 17.

For the large balanced clique enumeration problem, we implemented the baseline algorithm BCE and our advanced algorithm BCE* (Algorithm 19). In addition, we also implemented BCE-Adv, which is an improved version of BCE by (1) invoking MBC* to compute $\omega_\tau(G)$ and (2) conducting degree-based pruning and coloring-based upper bounding via ignoring edge signs.

For the polarization factor problem, we implemented PF-E (Section 5.5.1), PF-BS (Section 5.5.2) and PF* (Algorithm 20). In addition, we also implemented PF*-DOrder, which is the variant of PF* that replaces POrder with DOrder.

For the generalized maximum balanced clique problem, we implemented gMBC and gMBC*.

All the algorithms are implemented in C++, and all the experiments are conducted on a machine with an Intel Core-i7 3.20GHz CPU and Ubuntu system. The time cost is measured as the amount of wall-clock time elapsed during the program's execution. Unless otherwise specified, experiments are conducted with threshold $\tau = 3$ by default.

Datasets. We evaluate the algorithms on 14 datasets. Bitcoin, Reddit and Epinions are signed networks downloaded from SNAP [1]. AdjWordNet [2] captures the synonym-and-antonym relation among adjectives in the English language. Referendum and WikiConflict are signed networks used in (Bonchi et al., 2019) and are obtained from the authors of (Bonchi et al., 2019). Amazon, BookCross, TripAdvisor and YahooSong are rating networks downloaded from KONECT [3]. We transform them into signed graphs as follows. For each pair of users, if they have enough number of close (resp. opposite) rating scores to a set of items, we assign a positive (resp. negative) edge between them. DBLP is the signed network used in (Li et al., 2018a), which is downloaded from the *dblp database* [4] and processed in the same way as (Li et al., 2018a), i.e., assign "+" to an edge (u, v) if the number of papers co-authored by u and v is no less than 2, otherwise assign "−" to (u, v). Douban is a signed network used in (Chu et al., 2016). It is a movie-rating based social network in which an edge is negative if two users have very different movie preferences, and is positive otherwise. In addition, we also evaluate the algorithms on two synthetic datasets, SN1 and SN2, which are generated by the synthetic signed network generator SRN with default settings (Su et al., 2017). Statistics of the graphs are shown in Table 5.1, in which $\frac{|E^-|}{|E|}$ is the ratio of negative edges and $\omega_3(G)$ is the maximum balanced clique size

[1] https://snap.stanford.edu/
[2] https://wordnet.princeton.edu/
[3] http://konect.cc/networks/
[4] https://dblp.uni-trier.de/

under threshold $\tau = 3$.

Table 5.1: Statistics of datasets

Dataset	$\|V\|$	$\|E\|$	$\frac{\|E^-\|}{\|E\|}$	$\|C^*\|$	$\beta(G)$	Category
Bitcoin	5,881	21,492	0.15	11	5	Trade
AdjWordNet	16,259	76,845	0.32	60	28	Language
Reddit	54,075	220,151	0.08	8	3	Social
Referendum	10,884	251,406	0.05	19	5	Political
Epinions	131,828	711,210	0.17	15	6	Social
WikiConflict	116,717	2,026,646	0.63	6	3	Editing
Amazon	176,816	2,685,570	0.11	29	7	Rating
BookCross	63,535	3,890,104	0.07	550	118	Rating
DBLP	2,387,365	11,915,023	0.72	73	24	Coauthor
Douban	1,588,455	18,709,948	0.25	116	43	Social
TripAdvisor	145,315	20,569,277	0.14	1,916	201	Rating
YahooSong	1,000,990	30,139,524	0.18	127	21	Rating
SN1	2,000,000	50,154,048	0.41	13	5	Synthetic
SN2	2,000,000	111,573,268	0.39	19	7	Synthetic

5.7.1 Effectiveness of Maximum Balanced Clique

Compare with PolarSeeds (Xiao et al., 2020). We first compare the quality of maximum balanced clique computed by our algorithm MBC* with the polarized community computed by PolarSeeds (Xiao et al., 2020). As PolarSeeds has query vertices (i.e., seeds), we randomly pick 100 pairs of good seeds in the same way as (Xiao et al., 2020) and report the average result quality. Specifically, vertices u and v are considered to be seeds if $(u, v) \in E^-$, $d_G^+(u) > t$ and $d_G^+(v) > t$, where t is a pre-defined positive integer. The other parameters are set as their default values (i.e., $\epsilon = 10^{-3}$ and $\kappa = 0.9$). The result for the quality metric *Polarity* (Bonchi et al., 2019; Xiao et al., 2020) is shown in Figure 5.6. *Polarity* counts the number of edges that agree with the polarized structure and penalizes it by its size. That is, a polarized community of (Xiao et al., 2020) is represented as (C_1, C_2) which is similar to our C_L and C_R, and Polarity$(C_1, C_2) = \frac{|E^+(C_1) \cup E^+(C_2)| + 2|E^-(C_1, C_2)|}{|C_1 \cup C_2|}$, where $E^+(C)$ means the set of positive edges in C and $E^-(C_1, C_2)$ means the set of negative edges between C_1 and C_2. We can see that MBC* produces higher-quality results compared with PolarSeeds on all datasets. This is mainly because balanced cliques ensure that all edges agree with the polarized structure, and maximum balanced clique makes the Polarity even larger. Note that, besides Polarity, there are also other two quality metrics used in (Xiao et al., 2020), i.e., signed bipartiteness ratio (SBR) and harmonic mean of cohesion and opposition measures (HAM). It can be proved that the HAM of balanced cliques is always 1, the highest possible HAM value. Thus, MBC* also outperforms PolarSeeds regarding HAM. In terms of SBR which penalizes the edges going outside $C_1 \cup C_2$, PolarSeeds performs better than MBC*, since MBC* does not penalize the edges going outside $C_L \cup C_R$.

Case Studies. We conduct case studies for the maximum balaned cliques on datasets Reddit and AdjWordNet. In Reddit, each vertex represents a subreddit,

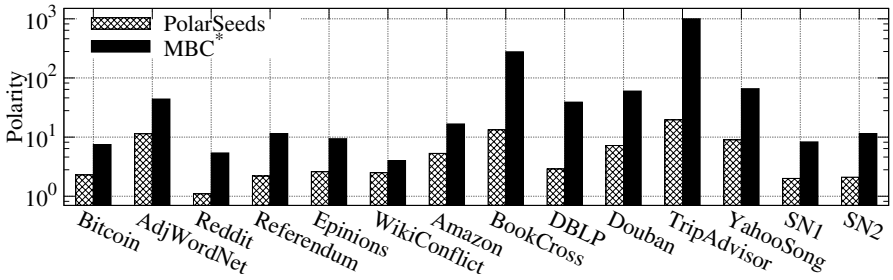


Figure 5.6: Polarity (the larger, the better)

Table 5.2: Case study on Reddit

C_L	C_R
videos, gaming, mma, thepopcornstand, canada	subredditdrama, truereddit-drama, drama

and positive and negative edges are created according to the sentiment between subreddits. The maximum balanced clique for $\tau = \beta(G) = 3$ as computed by MBC* is shown in Table 5.2. Subreddits in C_L usually interact with each other and share interesting posts about videos, games and pictures. All of them show negative sentiments to the subreddits drama in C_R. This shows that maximum balanced clique detects conflict groups in Reddit.

In AdjWordNet, each vertex represents an adjective, and a positive (resp. negative) edge indicates synonymous (resp. antonymous) relationship. The maximum balanced clique for $\tau = \beta(G) = 28$ has 60 vertices with $|C_L| = 28$ and $|C_R| = 32$, where a snippt is shown in Table 5.3. We can see that words in the same group, i.e., C_L or C_R, have similar meaning and words in different groups are mostly antonymous with each other. This case study shows that maximum balanced clique reveals interesting patterns in AdjWordNet.

We also run the MBCEnum algorithm of (Chen et al., 2020) to enumerate all maximal balanced cliques. For the above settings of τ, MBCEnum reports 197 cliques for Reddit and 1 clique for AdjWordNet. For Reddit, this huge number of results may overwhelm end-users, and moreover, most of the enumerated cliques (i.e., 93%) are of size only 6 and they heavily overlap with each other. In contrast, our algorithm efficiently detects the largest balanced clique, which can be regarded as the dominating polarized group. For AdjWordNet, the sole clique enumerated by MBCEnum is the same as the one computed by our algorithm MBC*, as one would expect. On the other hand, MBC* runs significantly faster than MBCEnum, e.g., 50× and 200× faster on Reddit and AdjWordNet, respectively.

5.7.2 Efficiency for Maximum Balanced Clique Computation

Against Enumeration-based Baseline Algorithm MBC. We first evaluate our algorithm MBC* against the enumeration-based baseline algorithm MBC. The results on all datasets for $\tau = 3$ are reported in Figure 5.7. We can clearly see that

Table 5.3: Case study on AdjWordNet

C_L	C_R
good, better, best, wonderful, excellent, great, superior, awesome, correct, well, sound, respectable, right, superb, honorable, optimum, terrific....	bad, worse, worst, terrible, poor, awful, inferior, unwell, weak, contemptible, dreadful, unsound, wrong, dishonorable, incorrect, unrespectable, horrific....

MBC* significantly outperforms MBC on all datasets, and the improvement is up-to three orders of magnitude (e.g., on Douban). This is due to our strategy of transforming the maximum balanced clique problem to a series of maximum dichromatic clique problems which remove edge signs and sparsify the graph. In Figure 5.7, we also report the running time of MBC-noER and MBC*-withER. We can see that EdgeReduction improves the efficiency for MBC (compared with MBC-noER), but degrades the performance for MBC*-withER (compared with MBC*). This is because MBC is very slow and thus EdgeReduction can improve the efficiency by reducing the graph instance. In contrast, MBC* runs very fast, and thus EdgeReduction incurs a large overhead for MBC* due to having a high time complexity. Thus, we omit MBC-noER and MBC*-withER in the remaining testings.

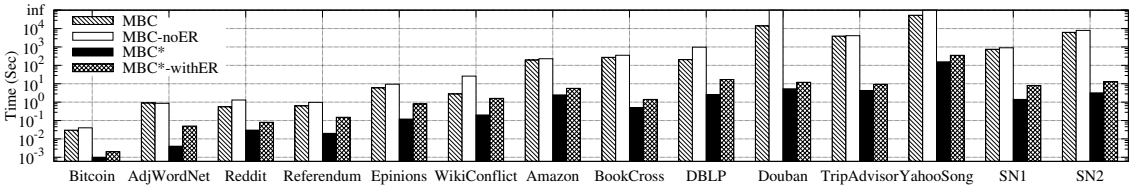
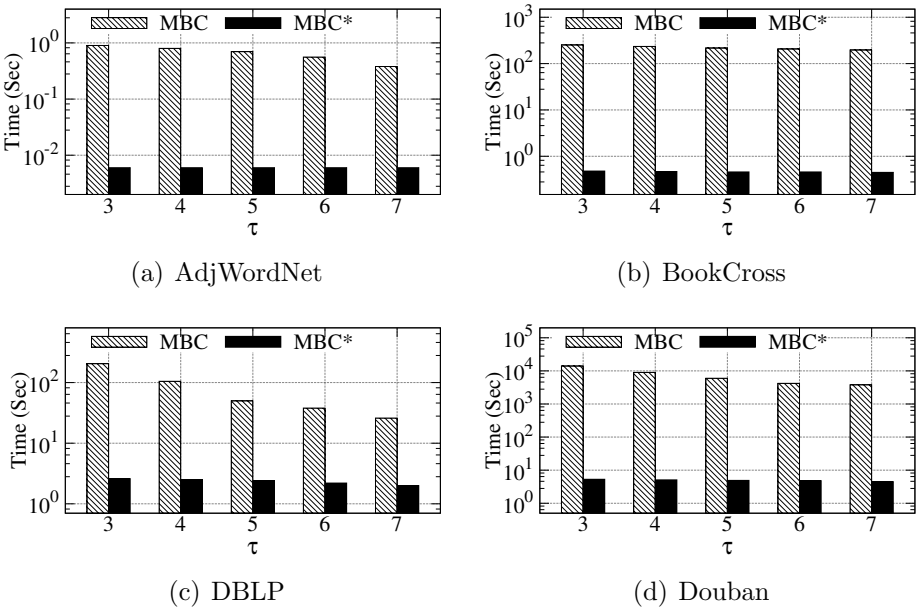


Figure 5.7: Running time on all graphs for maximum balanced clique computation ($\tau = 3$)

The results of evaluating MBC* against MBC by varying τ from 3 to 7 are shown in Figure 5.8. MBC* consistently outperforms MBC for different τ values. We also observe that the running time of MBC decreases as τ increases, while that of MBC* is almost insensitive to τ. This is because for larger τ values, the pruning power of EdgeReduction that is used in MBC strengthens. Nevertheless, MBC is still orders of magnitude slower than MBC* for $\tau = 7$.

Evaluate the Influence of Our MDC Transformation. In this experiment, we evaluate the influence of our MDC transformation — i.e., transform a maximum balanced clique problem to a series of maximum dichromatic clique problems — on the performance of our algorithm MBC*. We compare MBC* against MBC-Adv that does not apply the transformation but conducts the degree-based pruning and coloring-based bounding by simply discarding edge signs. From Figure 5.9, we can see that MBC* outperforms MBC-Adv by more than one order of magnitude. Thus, our MDC transformation improves the performance.

To get a deeper insight, we also evaluate the average size reduction brought by our MDC transformation. Recall that for each ego-network G_u, we first reduce it to g_u by removing all conflicting edges, and then reduce g_u to g by degree-based



(a) AdjWordNet (b) BookCross

(c) DBLP (d) Douban

Figure 5.8: Varying the threshold τ

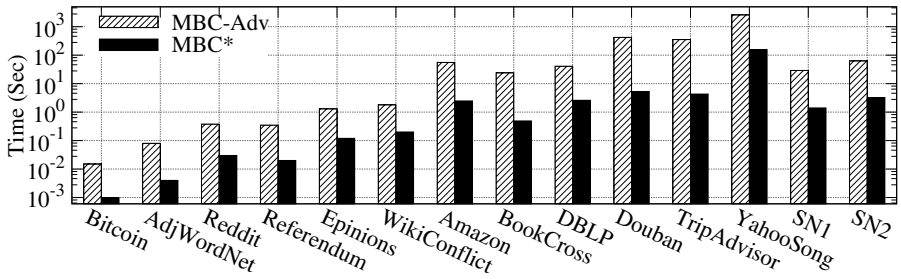


Figure 5.9: Influence of our MDC transformation

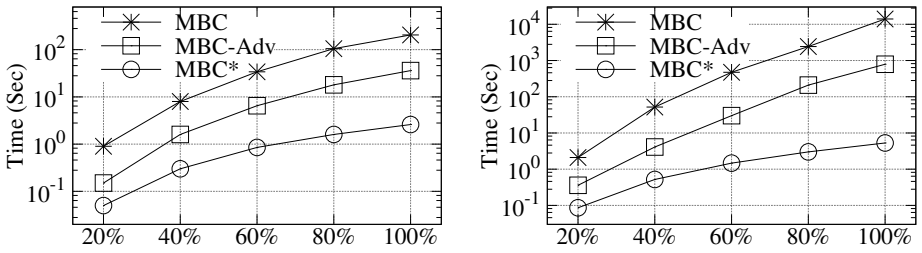
pruning (Line 7 of Algorithm 17). We use SR1 to denote the average size reduction ratio after stage 1 (i.e., SR1 $= 1 - \frac{|E(g_u)|}{|E(G_u)|}$), and use SR2 to denote the average size reduction ratio after stage 2 (i.e., SR2 $= 1 - \frac{|E(g)|}{|E(G_u)|}$). The results of SR1 and SR2 are shown in the fourth and fifth columns of Table 5.4, respectively. We see that around 50% edges are removed on average after stage 1, and almost 80% edges are removed on average after the two stages. This demonstrates that the subgraph sizes are significantly reduced before passing to MDC. We also report the number of MDC instances that are generated by our algorithm MBC* in the third column of Table 5.4. We see that the number of MDC instances is very small compared with $n = |V|$. This confirms the superiority of our MDC transformation. Note that for Bitcoin, AdjWordNet and TripAdvisor, the number of MDC instances is 0. This is because the heuristic solution found by MBC-Heu is guaranteed to be optimal, and thus no MDC instance is generated.

Scalability Testing. We now test the scalability of MBC, MBC-Adv and MBC* on two large datasets DBLP and Douban. We randomly sample vertices from 20% to 100% in the original graph. For each sampled vertex set, we obtain the induced

Table 5.4: Running statistics of MBC* and PF* for $\tau = 3$ (SR1 and SR2 are size reduction ratios, the larger the better)

Graphs	MBC*				PF*			
	Heu	#MDC	SR1	SR2	Heu	#DCC	SR1	SR2
Bitcoin	11	0	-	-	4	1	20%	20%
AdjWordNet	60	0	-	-	28	0	-	-
Reddit	6	96	41%	86%	2	14	40%	92%
Referendum	19	20	6%	31%	3	2	22%	22%
Epinions	11	82	30%	70%	5	10	35%	66%
WikiConflict	0	43	48%	94%	2	1	54%	80%
Amazon	15	952	64%	81%	6	825	73%	89%
BookCross	150	3	61%	85%	4	150	56%	90%
DBLP	31	26	49%	80%	2	23	51%	82%
Douban	83	10	73%	98%	27	6	72%	95%
TripAdvisor	1916	0	-	-	138	63	39%	49%
YahooSong	36	575	23%	31%	7	563	29%	39%
SN1	10	19	45%	90%	4	16	49%	92%
SN2	12	28	42%	85%	5	23	46%	89%

subgraph of the vertex set as the input data. As shown in Figure 5.10, the running time of all algorithms increases as the graph size increases. This is because a larger graph means a larger search space and thus a higher running time. Nevertheless, MBC* outperforms MBC and MBC-Adv in all the settings and scales better to the graph size.



(a) DBLP (b) Douban

Figure 5.10: Scalability testing ($\tau = 3$, vary graph size)

Evaluate Our Heuristic Algorithm. We now evaluate our heuristic algorithm MBC-Heu. The size of the balanced clique found by MBC-Heu, which is used as the initial clique in MBC*, is shown in the second column of Table 5.4. On Bitcoin, AdjWordNet, Referendum and TripAdvisor, MBC-Heu finds the optimal solution. On some of the graphs, the balanced clique found by MBC-Heu is far from optimal. For example, on BookCross, the balanced clique found by MBC-Heu is of size 150 while the maximum balanced clique size is 550. In the extreme case, MBC-Heu fails to find a valid initial solution on WikiConflict due to the constraint τ.

5.7.3 Efficiency for Large Balanced Clique Enumeration

Enumerating All Maximal Balanced Cliques. We first compare our algorithm BCE* with the maximal balanced clique enumeration algorithm MBCEnum proposed in (Chen et al., 2020), i.e., we set $\alpha = \infty$ for our problem. The results on all graphs are shown in Figure 5.11. Our algorithm BCE* consistently runs faster than MBCEnum, and the improvement can be up to two orders of magnitude, e.g., on graphs BookCross, Douban, TripAdvisor, YahooSong, SN1 and SN2. This demonstrates the superiority of our novel graph reduction technique that transforms maximal balanced clique enumeration over G to maximal dichromatic clique enumeration over small dichromatic networks of G.

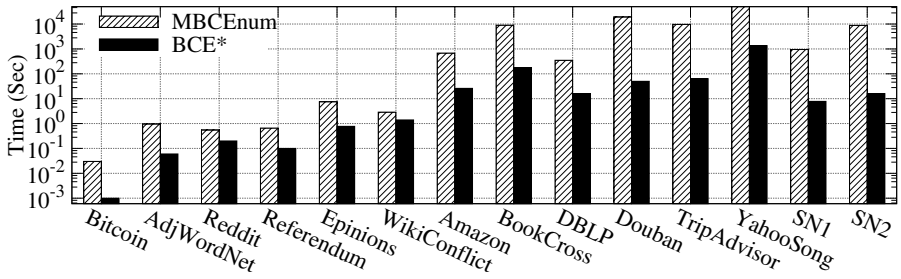


Figure 5.11: Compare with MBCEnum (Chen et al., 2020) for $\alpha = \infty$ ($\tau = 3$)

Against Baseline Algorithm BCE. In this evaluation, we compare our algorithm BCE* against the baseline algorithm BCE for enumerating large balanced cliques. The results on all datasets for $\tau = 3$ and $\alpha = 1$ are reported in Figure 5.12. We can clearly see that BCE* significantly outperforms BCE on all datasets, and the improvement is up-to four orders of magnitude (e.g., on BookCross). The efficiency of BCE* over BCE is mainly due to two reasons. Firstly, BCE* invokes our algorithm MBC* to efficiently obtain the maximum balanced clique C^*, based on which a large portion of the input graph can be pruned by Line 3 of Algorithm 19. Secondly, our novel graph reduction technique transforms large balanced clique enumeration problem over G to a series of large dichromatic clique enumeration problems over small dichromatic networks of G. The dichromatic networks are generally small after removing conflicting edges and pruning unpromising vertices based on λ and τ.

To separately evaluate the advantage of our graph reduction technique, we compare BCE* against BCE-Adv that also invokes MBC* for computing $\omega_\tau(G)$ but does not apply our graph reduction technique. In contrast, BCE-Adv conducts the degree-based pruning and coloring-based upper bounding by simply discarding edge signs. The results are also shown in Figure 5.12. We can see that BCE* consistently outperforms BCE-Adv, and the improvement is up to one order of magnitude, e.g., on graphs Amazon, BookCross, DBLP and Douban. This demonstrates the advantage of our graph reduction technique.

Evaluate the Algorithms by Varying α. In this experiment, we evaluate the algorithms BCE*, BCE and BCE-Adv by varying α. The results on graphs Epinions, Amazon, BookCross, and Douban are shown in Figure 5.13. Note that the values

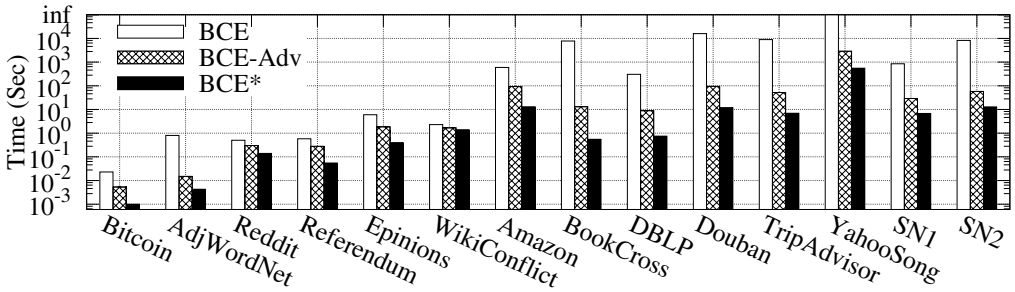
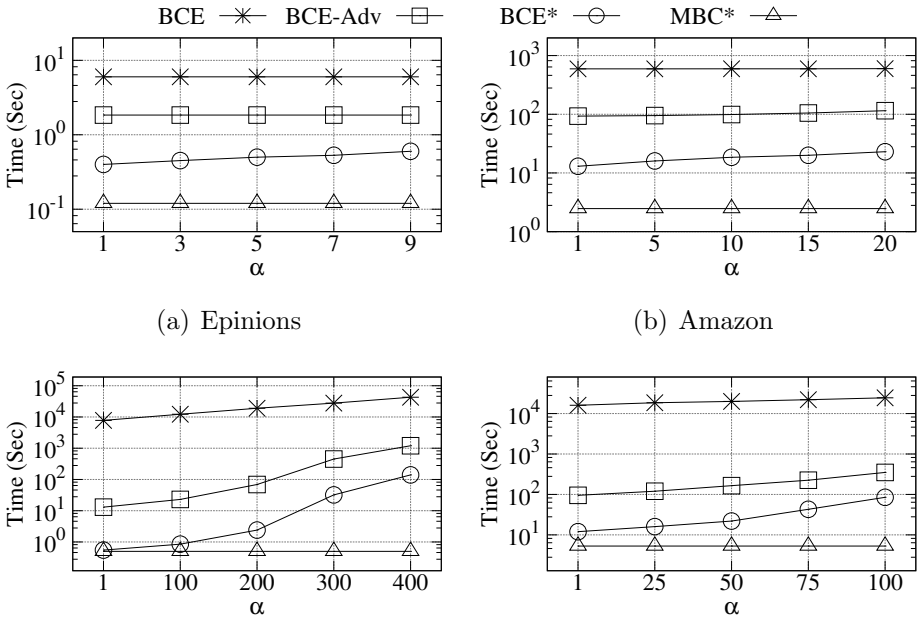


Figure 5.12: Running time on all graphs for large balanced clique enumeration ($\tau = 3$ and $\alpha = 1$)

of α are selected according to $\omega_\tau(G)$ and thus are different for different graphs; please refer to Table 5.1 for the values of $\omega_\tau(G)$ for these graphs. We can see that as expected, the running time of all the three algorithms increase, since more results will be reported. For example, on BookCross, the number of large maximal balanced cliques increases from 384 to 19,841,800 when α increases from 1 to 400. Nevertheless, BCE* consistently outperforms both BCE and BCE-Adv. In Figure 5.13 we also report the running time of MBC* which is invoked by BCE* and BCE-Adv to compute $\omega_\tau(G)$ and is irrelevant to α. We can see that the running time of MBC* is not significant compared with that of BCE*; this is more evident for large α. Thus, it makes sense to invoke MBC* for computing $\omega_\tau(G)$ in BCE*.



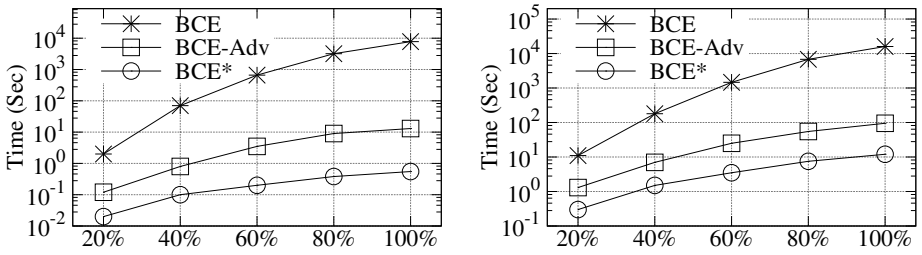
(a) Epinions (b) Amazon

(c) BookCross (d) Douban

Figure 5.13: Running time by varying α ($\tau = 3$)

Scalability Testing. In this experiment, we test the scalability of BCE, BCE-Adv and BCE* on BookCross and Douban by varying the graph size to be $\{20\%, \ldots, 100\%\}$

of the input graph. The results are shown in Figure 5.14. We can see that when the graph size increases, the processing time of all algorithms increases which is as expected. Nevertheless, BCE* outperforms BCE and BCE-Adv in all cases and scales better than BCE.



(a) BookCross (b) Douban

Figure 5.14: Scalability testing (vary graph size)

The Number of Large Maximal Balanced Cliques by Varying α. In this experiment, we vary α from 0 to 9, and report the number of large maximal balanced cliques that are of size at least $\omega_\tau(G) - \alpha$. We also report the number of large maximal balanced cliques when $\alpha = |V|$. In this case, all maximal balanced cliques will be enumerated. The results on all graphs are shown in Table 5.5. As expected, the number of reported balanced cliques increases when α becomes larger. We can also see that the number of maximal balanced cliques of size $\omega_\tau(G) - \alpha$ is already very large for $\alpha = 9$, e.g., on Referendum, Epinions, Amazon, Douban, and YahooSong; the number of all maximal balanced cliques will be even larger when $\alpha = |V|$. This demonstrates the usefulness of our model which only enumerates large maximal balanced cliques: by varying α, end-users can control the number of reported balanced cliques; also, we guarantee that all non-reported maximal balanced cliques will be of smaller size than the reported ones.

Table 5.5: Number of large balanced cliques by varying α ($\tau = 3$)

Graphs	α						
	0	1	3	5	7	9	$\lvert V\rvert$
Bitcoin	6	31	87	133	133	133	133
AdjWordNet	2	4	4	4	4	4	858
Reddit	2	14	197	197	197	197	197
Referendum	2	27	585	2,661	4,773	5,958	6,773
Epinions	9	86	767	3,182	9,717	17,574	17,574
WikiConflict	37	37	37	37	37	37	37
Amazon	43	572	5,471	22,268	66,611	155,155	1,556,500
BookCross	192	384	384	384	384	384	20,643,998
DBLP	2	4	5	5	5	5	32,213
Douban	83	210	773	1,298	3,768	6,293	93,275
TripAdvisor	2	2	2	2	2	2	8,570
YahooSong	3	17	83	312	767	1,563	195,964
SN1	1	5	14	29	56	56	56
SN2	1	7	18	35	65	133	397

5.7.4 Efficiency for the Polarization Factor Problem

Against PF-E, PF-BS and PF*-DOrder. Figure 5.15 reports the running time of PF-E, PF-BS, PF*-DOrder and PF* on all the graphs. We can see that PF* outperforms the enumeration-based baseline algorithm PF-E by several orders of magnitude on all the graphs. For example, on Douban, PF* computes the polarization factor in 5 seconds, while PF-E takes more than 4 hours. Our binary search-based algorithm PF-BS is significantly faster than PF-E, due to our efficient algorithm MBC* for verifying τ. However, as PF-BS needs to invoke MBC* multiple times, it is almost one order of magnitude slower than PF*. This demonstrates the superiority of PF* that transforms the polarization factor problem to a series of dichromatic clique checking problems.

In Figure 5.15, we also report the running time of PF*-DOrder, which is the variant of PF* that uses the degeneracy ordering DOrder instead of the polarization ordering. We see that PF*-DOrder is slower than PF*. This demonstrates the superiority of the polarization ordering for computing the polarization factor.

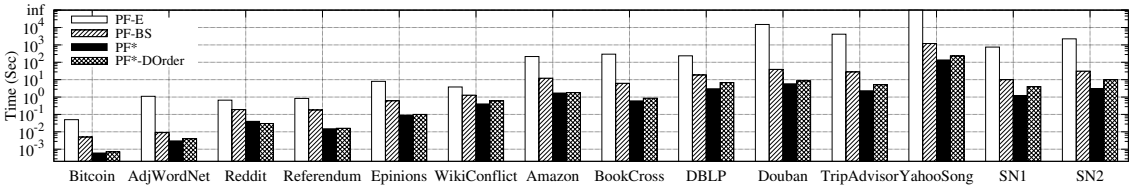


Figure 5.15: Running time on all graphs for polarization factor

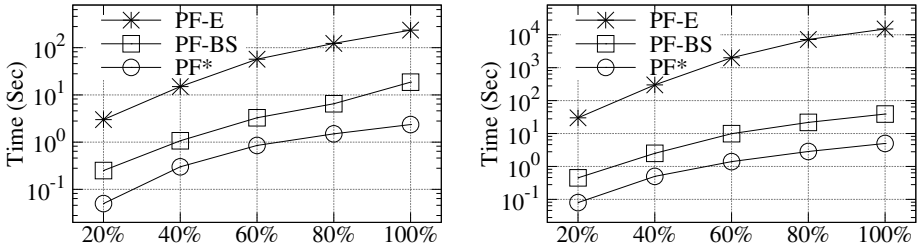
Evaluate the Heuristic Algorithm and DCC Transformation. In this experiment, we report the running statistics of PF*. The initial lower bound of the polarization factor $\beta(G)$ computed by MBC-Heu is shown in the sixth column of Table 5.4. We see that MBC-Heu obtains the exact value of $\beta(G)$ on AdjWordNet. On some of the graphs, the initial lower bound of $\beta(G)$ is much smaller than the exact value. For example, on BookCross, the initial lower bound is 4 while the exact $\beta(G)$ is 118.

The number of DCC instances generated by PF* and the average instance size reduction ratio are shown in last three columns of Table 5.4, which exhibit similar phenomenon as MBC*. That is, the number of DCC instances is very small compared with $n = |V|$, and the transformation leads to a significant reduction on each dichromatic-network. This confirms the superiority of our DCC transformation.

Scalability Testing. In this experiment, we test the scalability of PF-E, PF-BS and PF* on DBLP and Douban by varying their vertices from 20% to 100%. As shown in Figure 5.16, when the graph scales up, the processing time of all algorithms increases as well because of the increasing search space. However, PF* outperforms other algorithms in all cases and scales well to the graph size.

5.7.5 The Generalized Maximum Balanced Clique Problem

In this subsection, we evaluate the algorithms for computing a maximum balanced clique for each $0 \leq \tau \leq \beta(G)$. Firstly, we observe that the number of distinct max-



(a) DBLP (b) Douban

Figure 5.16: Scalability testing (vary graph size)

imum balanced clique in $\{C^0, C^1, \ldots, C^{\beta(G)}\}$, where C^i is the maximum balanced clique for constraint $\tau = i$, is much smaller than $\beta(G)+1$. For example, the number is 39 for BookCross which has $\beta(G) = 118$, and the number is 14 for TripAdvisor which has $\beta(G) = 201$; the number of such cliques for all datasets is shown in the second column of Table 5.6, denoted by $|\mathbb{C}|$. This is because the maximum balanced clique is the same for many of the different τ values. In the third column of Table 5.6, we also report the size of the maximum balanced clique for $\tau = \beta(G)$ and that for $\tau = 0$; the two numbers in the subscripts represent the sizes of the two sides of the clique, i.e., $|C_L|$ and $|C_R|$. For example, for the dataset BookCross, the maximum balanced clique for $\tau = \beta(G) = 118$ has 240 vertices with $|C_L| = 118$ and $|C_R| = 122$, and the maximum balanced clique for $\tau = 0$ has 614 vertices with $|C_L| = 1$ and $|C_R| = 613$. We can see that C^0 is highly skewed while $C^{\beta(G)}$ is well balanced, in terms of the sizes of C_L and C_R. As the number of distinct maximum balanced cliques for all possible values of the threshold τ is not large, end-users can easily go through all these maximum balanced cliques to find the ones that suit their needs.

Table 5.6: Distinct number of maximum balanced cliques for different τ values (i.e., $|\mathbb{C}| = |\{C^0, C^1, \ldots, C^{\beta(G)}\}|$)

Graph	$\lvert\mathbb{C}\rvert$	Size Range of the Cliques
Bitcoin	2	$10_{\langle 5\rvert 5\rangle} \rightarrow 11_{\langle 4\rvert 7\rangle}$
AdjWordNet	1	$60_{\langle 28\rvert 32\rangle} \rightarrow 60_{\langle 28\rvert 32\rangle}$
Reddit	4	$8_{\langle 3\rvert 5\rangle} \rightarrow 17_{\langle 0\rvert 17\rangle}$
Referendum	6	$17_{\langle 5\rvert 12\rangle} \rightarrow 35_{\langle 0\rvert 35\rangle}$
Epinions	7	$12_{\langle 6\rvert 6\rangle} \rightarrow 93_{\langle 0\rvert 93\rangle}$
WikiConflict	4	$6_{\langle 3\rvert 3\rangle} \rightarrow 16_{\langle 0\rvert 16\rangle}$
Amazon	8	$15_{\langle 7\rvert 8\rangle} \rightarrow 42_{\langle 0\rvert 42\rangle}$
BookCross	39	$240_{\langle 118\rvert 122\rangle} \rightarrow 614_{\langle 1\rvert 613\rangle}$
DBLP	7	$49_{\langle 24\rvert 25\rangle} \rightarrow 247_{\langle 1\rvert 246\rangle}$
Douban	13	$88_{\langle 43\rvert 45\rangle} \rightarrow 139_{\langle 0\rvert 139\rangle}$
TripAdvisor	14	$448_{\langle 201\rvert 247\rangle} \rightarrow 1916_{\langle 45\rvert 1871\rangle}$
YahooSong	10	$43_{\langle 21\rvert 22\rangle} \rightarrow 353_{\langle 0\rvert 353\rangle}$
SN1	6	$10_{\langle 5\rvert 5\rangle} \rightarrow 19_{\langle 0\rvert 19\rangle}$
SN2	8	$15_{\langle 7\rvert 8\rangle} \rightarrow 24_{\langle 0\rvert 24\rangle}$

The running time of algorithms gMBC and gMBC* on all the datasets is shown in Figure 5.17. Recall that both algorithms invoke MBC*. The increasing factor of

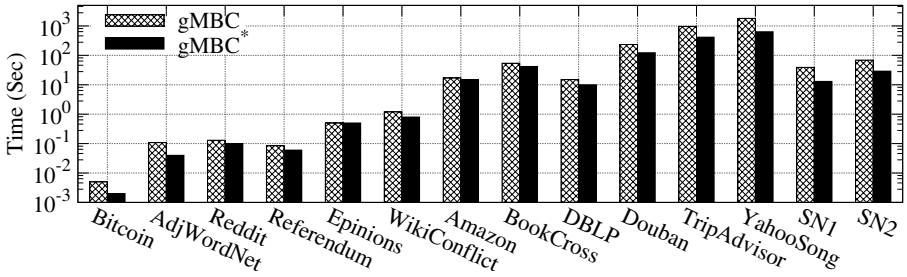


Figure 5.17: Running time of gMBC and gMBC* on all graphs

gMBC and gMBC* over MBC* is generally related to the polarization factor $\beta(G)$ of the graph, since the larger $\beta(G)$, the more rounds they need to invoke MBC*. Nevertheless, gMBC* consistently runs faster than gMBC due to the computation sharing in the former.

5.8 Chapter Summary

In this chapter, we first studied the maximum balanced clique problem in signed graphs. We proposed techniques to transform the maximum balanced clique problem over G to a series of maximum dichromatic clique problems over small subgraphs of G. The transformation not only removes edge signs (and thus the structural balanced constraint) but also sparsifies the edge set. In addition, we also extended our techniques to the large balanced clique enumeration problem, the polarization factor problem, and the problem of reporting a maximum balanced clique for each $\tau \geq 0$. Experimental results on real datasets demonstrated the efficiency, effectiveness and scalability of our algorithms.

Chapter 6

Epilogue

In this thesis, we focus on cohesive subgraph identification in large graphs. The main contributions of this thesis are:

- *Formulation of size-bounded community and efficient algorithms for size-bounded community search.* We formulated and studied the size-bounded community search problem, which has many applications in the real-world. We proposed an efficient branch-reduce-and-bound algorithm for this problem by developing nontrivial reducing techniques, upper bounding techniques, and branching techniques. We conducted extensive experiments on large real graphs to demonstrate the efficiency and effectiveness of our algorithms.

- *Formulation of similar-biclique model and efficient algorithms for the maximal similar-biclique enumeration.* We formulated a novel cohesive subgraph model in bipartite graphs, i.e., similar-biclique, which demonstrated its superiorities compared with other models (e.g., in the task of anomaly detection). We developed a backtracking algorithm to enumerate all maximal similar-bicliques in a bipartite graph. Besides, we designed a novel index structure to facilitate the computation. We conducted extensive experiments on large real bipartite graphs to demonstrate the effectiveness of our model and the efficiency of our algorithms.

- *Efficient algorithms for the maximum balanced clique computation problem and its variants.* We proposed an efficient algorithm for the maximum balanced clique computation in signed graphs. A novel graph reduction technique was proposed to facilitate the computation. We also extended our techniques to the large balanced clique enumeration problem, the polarization factor problem and the generalized maximum balanced clique problem. We conducted extensive experiments on large real signed graphs to demonstrate the efficiency of our algorithms.

There are still some open problems that need further research.

- *Efficiency of size-bounded community search.* Due to the NP-hardness of size-bounded community search, current algorithms can only handle the queries with a relatively small size upper bound (i.e., h is usually less than 20). It

is desirable to design more advanced techniques to improve the efficiency of size-bounded community search.

- *More cohesiveness metrics.* There are many other cohesiveness metrics that have not been used for cohesive subgraph identification in signed graphs, such as quasi-clique, k-plex, and k-core. Thus, it would be interesting to combine structure balance theory with other cohesiveness metrics. Besides, it is also possible to integrate similarity constraints into other dense bipartite subgraph models, such as k-biplex and (α, β)-core, in a similar way to similar-biclique.

- *Other types of graphs.* In recent years, many novel network models have been developed, such as public-private networks, uncertain graphs and heterogeneous information networks. Thus, it would be interesting to formulate novel cohesive subgraph models in these types of graphs.

Bibliography

Abello, J., Resende, M. G., and Sudarsky, S. (2002). Massive quasi-clique detection. In *Latin American symposium on theoretical informatics*, pages 598–612. Springer.

Abidi, A., Zhou, R., Chen, L., and Liu, C. (2021). Pivot-based maximal biclique enumeration. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 3558–3564.

Ahmed, M., Mahmood, A. N., and Islam, M. R. (2016). A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55:278–288.

Alexe, G., Alexe, S., Crama, Y., Foldes, S., Hammer, P. L., and Simeone, B. (2004). Consensus algorithms for the generation of all maximal bicliques. *Discrete Applied Mathematics*, 145(1):11–21.

Allahbakhsh, M., Ignjatovic, A., Benatallah, B., Beheshti, S.-M.-R., Bertino, E., and Foo, N. (2013). Collusion detection in online rating systems. In *Asia-Pacific Web Conference*, pages 196–207. Springer.

Bader, D. A. and Madduri, K. (2006). Gtgraph: A synthetic graph generator suite. *Atlanta, GA, February*, 38.

Batagelj, V. and Zaversnik, M. (2003). An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*.

Bedi, P. and Sharma, C. (2016). Community detection in social networks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 6(3):115–135.

Bi, F., Chang, L., Lin, X., and Zhang, W. (2018). An optimal and progressive approach to online search of top-k influential communities. *Proceedings of the VLDB Endowment*, 11(9):1056–1068.

Boldi, P. and Vigna, S. (2004). The webgraph framework i: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602.

Bonchi, F., Galimberti, E., Gionis, A., Ordozgoiti, B., and Ruffo, G. (2019). Discovering polarized communities in signed networks. In *Proceedings of the 28th acm international conference on information and knowledge management*, pages 961–970.

Bron, C. and Kerbosch, J. (1973). Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577.

Chang, L. (2019). Efficient maximum clique computation over large sparse graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 529–538.

Chang, L. (2020). Efficient maximum clique computation and enumeration over large sparse graphs. *VLDB Journal International Journal on Very Large Data Bases*, 29(5).

Chang, L., Li, W., Qin, L., Zhang, W., and Yang, S. (2017). pscan: Fast and exact structural graph clustering. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):387–401.

Chang, L., Lin, X., Qin, L., Yu, J. X., and Zhang, W. (2015). Index-based optimal algorithms for computing steiner components with maximum connectivity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 459–474.

Chang, L. and Qin, L. (2019). Cohesive subgraph computation over large sparse graphs. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 2068–2071. IEEE.

Chang, L., Yu, J. X., and Qin, L. (2013a). Fast maximal cliques enumeration in sparse graphs. *Algorithmica*, 66(1):173–186.

Chang, L., Yu, J. X., Qin, L., Lin, X., Liu, C., and Liang, W. (2013b). Efficiently computing k-edge connected components via graph decomposition. In *Proceedings of the 2013 ACM SIGMOD international conference on management of data*, pages 205–216.

Chen, C. C., Wan, Y.-H., Chung, M.-C., and Sun, Y.-C. (2013). An effective recommendation method for cold start new users using trust and distrust networks. *Information Sciences*, 224:19–36.

Chen, L., Liu, C., Zhou, R., Li, J., Yang, X., and Wang, B. (2018). Maximum co-located community search in large scale social networks. *Proceedings of the VLDB Endowment*, 11(10):1233–1246.

Chen, L., Liu, C., Zhou, R., Xu, J., and Li, J. (2021a). Efficient exact algorithms for maximum balanced biclique search in bipartite graphs. In *Proceedings of the 2021 International Conference on Management of Data*, pages 248–260.

Chen, L., Liu, C., Zhou, R., Xu, J., and Li, J. (2022). Efficient maximal biclique enumeration for large sparse bipartite graphs. *Proceedings of the VLDB Endowment*, 15(8):1559–1571.

Chen, Y., Zhang, J., Fang, Y., Cao, X., and King, I. (2021b). Efficient community search over large directed graphs: An augmented index-based approach. In

Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, pages 3544–3550.

Chen, Z., Yuan, L., Lin, X., Qin, L., and Yang, J. (2020). Efficient maximal balanced clique enumeration in signed networks. In *Proceedings of The Web Conference 2020*, pages 339–349.

Chu, L., Wang, Z., Pei, J., Wang, J., Zhao, Z., and Chen, E. (2016). Finding gangs in war from signed networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1505–1514.

Clauset, A. (2005). Finding local community structure in networks. *Physical review E*, 72(2):026132.

Cohen, J. (2008). Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report*, 16(3.1).

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2022). *Introduction to algorithms*. MIT press.

Cui, W., Xiao, Y., Wang, H., Lu, Y., and Wang, W. (2013). Online search of overlapping communities. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*, pages 277–288.

Cui, W., Xiao, Y., Wang, H., and Wang, W. (2014). Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of Data*, pages 991–1002.

Dave, V., Guha, S., and Zhang, Y. (2013). Viceroi: Catching click-spam in search ad networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 765–776.

Dawande, M., Keskinocak, P., Swaminathan, J. M., and Tayur, S. (2001). On bipartite and multipartite clique problems. *Journal of Algorithms*, 41(2):388–403.

de Berg, M., Cheong, O., van Kreveld, M. J., and Overmars, M. H. (2008). *Computational geometry: algorithms and applications, 3rd Edition*. Springer.

Decelle, A., Krzakala, F., Moore, C., and Zdeborová, L. (2011). Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E*, 84(6):066106.

Ding, D., Li, H., Huang, Z., and Mamoulis, N. (2017). Efficient fault-tolerant group recommendation using alpha-beta-core. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 2047–2050.

Easley, D. A. and Kleinberg, J. M. (2010). *Networks, Crowds, and Markets - Reasoning About a Highly Connected World*. Cambridge University Press.

EL BACHA, R. and Zin, T. T. (2018). Ranking of influential users based on user-tweet bipartite graph. In *2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, pages 97–101. IEEE.

Eppstein, D. (1994). Arboricity and bipartite subgraph listing algorithms. *Information processing letters*, 51(4):207–211.

Eppstein, D., Löffler, M., and Strash, D. (2013). Listing all maximal cliques in large sparse real-world graphs. *Journal of Experimental Algorithmics (JEA)*, 18:3–1.

Erdős, P., Pach, J., Pollack, R., and Tuza, Z. (1989). Radius, diameter, and minimum degree. *Journal of Combinatorial Theory, Series B*, 47(1):73–79.

Fang, Y., Cheng, R., Li, X., Luo, S., and Hu, J. (2017). Effective community search over large spatial graphs. *Proceedings of the VLDB Endowment*, 10(6):709–720.

Fang, Y., Cheng, R., Luo, S., and Hu, J. (2016). Effective community search for large attributed graphs. *Proceedings of the VLDB Endowment*, 9(12):1233–1244.

Fang, Y., Huang, X., Qin, L., Zhang, Y., Zhang, W., Cheng, R., and Lin, X. (2020a). A survey of community search over big graphs. *The VLDB Journal*, 29(1):353–392.

Fang, Y., Wang, Z., Cheng, R., Wang, H., and Hu, J. (2018). Effective and efficient community search over large directed graphs. *IEEE Transactions on Knowledge and Data Engineering*, 31(11):2093–2107.

Fang, Y., Yang, Y., Zhang, W., Lin, X., and Cao, X. (2020b). Effective and efficient community search over large heterogeneous information networks. *Proceedings of the VLDB Endowment*, 13(6):854–867.

Figueiredo, R. and Frota, Y. (2014). The maximum balanced subgraph of a signed graph: Applications and solution approaches. *European Journal of Operational Research*, 236(2):473–487.

Gangireddy, S. C. R., Long, C., and Chakraborty, T. (2020). Unsupervised fake news detection: A graph-based approach. In *Proceedings of the 31st ACM conference on hypertext and social media*, pages 75–83.

Giatsidis, C., Cautis, B., Maniu, S., Thilikos, D. M., and Vazirgiannis, M. (2014). Quantifying trust dynamics in signed graphs, the s-cores approach. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 668–676. SIAM.

Goldberg, A. V. (1984). Finding a maximum density subgraph.

Hao, F., Yau, S. S., Min, G., and Yang, L. T. (2014). Detecting k-balanced trusted cliques in signed social networks. *IEEE internet computing*, 18(2):24–31.

Harary, F. et al. (1953). On the notion of balance of a signed graph. *The Michigan Mathematical Journal*, 2(2):143–146.

Hochbaum, D. S. (1996). Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. In *Approximation algorithms for NP-hard problems*, pages 94–143.

Hochbaum, D. S. (1998). Approximating clique and biclique problems. *Journal of Algorithms*, 29(1):174–200.

Hooi, B., Song, H. A., Beutel, A., Shah, N., Shin, K., and Faloutsos, C. (2016). Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 895–904.

Huang, X., Cheng, H., Qin, L., Tian, W., and Yu, J. X. (2014). Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1311–1322.

Huang, X. and Lakshmanan, L. V. (2017). Attribute-driven community search. *Proceedings of the VLDB Endowment*, 10(9):949–960.

Huang, X., Lakshmanan, L. V., and Xu, J. (2019). Community search over big graphs. *Synthesis Lectures on Data Management*, 14(6):1–206.

Huang, X., Lakshmanan, L. V., Yu, J. X., and Cheng, H. (2015). Approximate closest community search in networks. *Proceedings of the VLDB Endowment*, 9(4):276–287.

Inokuchi, A., Washio, T., and Motoda, H. (2000). An apriori-based algorithm for mining frequent substructures from graph data. In *European conference on principles of data mining and knowledge discovery*, pages 13–23. Springer.

Jiang, M., Cui, P., Beutel, A., Faloutsos, C., and Yang, S. (2014). Catchsync: catching synchronized behavior in large directed graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 941–950.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer.

Kim, J., Luo, S., Cong, G., and Yu, W. (2022). Dmcs: Density modularity based community search. *arXiv preprint arXiv:2204.07720*.

Kloster, K., Sullivan, B. D., and van der Poel, A. (2019). Mining maximal induced bicliques using odd cycle transversals. In *Proceedings of the 2019 SIAM International Conference on Data Mining, SDM 2019, Calgary, Alberta, Canada, May 2-4, 2019*, pages 324–332.

Knoke, D. and Yang, S. (2019). *Social network analysis*, volume 154. Sage Publications.

Krishnan, A., Deepak, P., Ranu, S., and Mehta, S. (2018). Leveraging semantic resources in diversified query expansion. *World Wide Web*, 21(4):1041–1067.

Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. (1999). Trawling the web for emerging cyber-communities. *Computer networks*, 31(11-16):1481–1493.

Kumar, S., Hamilton, W. L., Leskovec, J., and Jurafsky, D. (2018). Community interaction and conflict on the web. In *Proceedings of the 2018 world wide web conference*, pages 933–943.

Kumar, V., Joshi, N., Mukherjee, A., Ramakrishnan, G., and Jyothi, P. (2019). Cross-lingual training for automatic question generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4863–4872.

Kunegis, J. (2013). Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on world wide web*, pages 1343–1350.

Kunegis, J., Lommatzsch, A., and Bauckhage, C. (2009). The slashdot zoo: mining a social network with negative edges. In *Proceedings of the 18th international conference on World Wide Web*, pages 741–750.

Kuznetsov, S. O. (2001). On computing the size of a lattice and related decision problems. *Order*, 18(4):313–321.

Lee, V. E., Ruan, N., Jin, R., and Aggarwal, C. (2010). A survey of algorithms for dense subgraph discovery. In *Managing and mining graph data*, pages 303–336. Springer.

Lehmann, S., Schwartz, M., and Hansen, L. K. (2008). Biclique communities. *Physical review E*, 78(1):016108.

Leskovec, J., Huttenlocher, D., and Kleinberg, J. (2010). Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World Wide Web*, pages 641–650.

Leskovec, J. and Krevl, A. (2014). Snap datasets: Stanford large network dataset collection.

Lewis, H. R. (1983). Michael r. πgarey and david s. johnson. computers and intractability. a guide to the theory of np-completeness. wh freeman and company, san francisco1979, x+ 338 pp. *The Journal of Symbolic Logic*, 48(2):498–500.

Ley, M. (2002). The dblp computer science bibliography: Evolution, research issues, perspectives. In *International symposium on string processing and information retrieval*, pages 1–10. Springer.

Li, C., Zhang, F., Zhang, Y., Qin, L., Zhang, W., and Lin, X. (2019). Efficient progressive minimum k-core search. *Proceedings of the VLDB Endowment*, 13(3):362–375.

Li, C.-M., Fang, Z., and Xu, K. (2013). Combining maxsat reasoning and incremental upper bound for the maximum clique problem. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 939–946. IEEE.

Li, C.-M., Jiang, H., and Manyà, F. (2017). On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research*, 84:1–15.

Li, J., Li, H., Soh, D., and Wong, L. (2005). A correspondence between maximal complete bipartite subgraphs and closed patterns. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 146–156. Springer.

Li, J., Liu, G., Li, H., and Wong, L. (2007). Maximal biclique subgraphs and closed pattern pairs of the adjacency matrix: A one-to-one correspondence and mining algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1625–1637.

Li, L., Zhao, Y., Li, Y., Wahab, F., and Wang, Z. (2022). The most active community search in large temporal graphs. *Knowledge-Based Systems*, page 109101.

Li, R.-H., Dai, Q., Qin, L., Wang, G., Xiao, X., Yu, J. X., and Qiao, S. (2018a). Efficient signed clique search in signed networks. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 245–256. IEEE.

Li, R.-H., Qin, L., Yu, J. X., and Mao, R. (2015). Influential community search in large networks. *Proceedings of the VLDB Endowment*, 8(5):509–520.

Li, R.-H., Su, J., Qin, L., Yu, J. X., and Dai, Q. (2018b). Persistent community search in temporal networks. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 797–808. IEEE.

Liben-Nowell, D. and Kleinberg, J. (2007). The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031.

Liu, G., Sim, K., and Li, J. (2006). Efficient mining of large maximal bicliques. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 437–448. Springer.

Liu, Q., Zhao, M., Huang, X., Xu, J., and Gao, Y. (2020). Truss-based community search over large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2183–2197.

Liu, X., Li, J., and Wang, L. (2008). Quasi-bicliques: Complexity and binding pairs. In *International Computing and Combinatorics Conference*, pages 255–264. Springer.

Lyu, B., Qin, L., Lin, X., Zhang, Y., Qian, Z., and Zhou, J. (2020). Maximum biclique search at billion scale. *Proceedings of the VLDB Endowment*.

Ma, Y.-L., Yuan, Y., Zhu, F.-D., Wang, G.-R., Xiao, J., and Wang, J.-Z. (2019). Who should be invited to my party: A size-constrained k-core problem in social networks. *Journal of Computer Science and Technology*, 34(1):170–184.

Manchanda, P., Packard, G., and Pattabhiramaiah, A. (2015). Social dollars: The economic impact of customer participation in a firm-sponsored online customer community. *Marketing Science*, 34(3):367–387.

Martínez, V., Berzal, F., and Cubero, J.-C. (2016). A survey of link prediction in complex networks. *ACM computing surveys (CSUR)*, 49(4):1–33.

Maslov, E., Batsyn, M., and Pardalos, P. M. (2014). Speeding up branch and bound algorithms for solving the maximum clique problem. *Journal of Global Optimization*, 59(1):1–21.

Matula, D. W. and Beck, L. L. (1983). Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427.

Megiddo, N., Zemel, E., and Hakimi, S. L. (1983). The maximum coverage location problem. *SIAM Journal on Algebraic Discrete Methods*, 4(2):253–261.

Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Mokken, R. J. et al. (1979). Cliques, clubs and clans. *Quality & Quantity*, 13(2):161–173.

Ordozgoiti, B., Matakos, A., and Gionis, A. (2020). Finding large balanced subgraphs in signed networks. In *Proceedings of The Web Conference 2020*, pages 1378–1388.

Ou-Yang, L., Dai, D.-Q., and Zhang, X.-F. (2015). Detecting protein complexes from signed protein-protein interaction networks. *IEEE/ACM transactions on computational biology and bioinformatics*, 12(6):1333–1344.

Peeters, R. (2003). The maximum edge biclique problem is np-complete. *Discrete Applied Mathematics*, 131(3):651–654.

Sanderson, M. J., Driskell, A. C., Ree, R. H., Eulenstein, O., and Langley, S. (2003). Obtaining maximal concatenated phylogenetic data sets from large sequence databases. *Molecular biology and evolution*, 20(7):1036–1042.

Sarıyüce, A. E. and Pinar, A. (2018). Peeling bipartite networks for dense subgraph discovery. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 504–512.

Satuluri, V., Parthasarathy, S., and Ruan, Y. (2011). Local graph sparsification for scalable clustering. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, pages 721–732.

Scott, J. (1988). Social network analysis. *Sociology*, 22(1):109–127.

Seidman, S. B. (1983). Network structure and minimum degree. *Social networks*, 5(3):269–287.

Seidman, S. B. and Foster, B. L. (1978). A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154.

Shaham, E., Yu, H., and Li, X.-L. (2016). On finding the maximum edge biclique in a bipartite graph: a subspace clustering approach. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 315–323. SIAM.

Shahinpour, S., Shirvani, S., Ertem, Z., and Butenko, S. (2017). Scale reduction techniques for computing maximum induced bicliques. *Algorithms*, 10(4):113.

She, J., Tong, Y., Chen, L., and Cao, C. C. (2016). Conflict-aware event-participant arrangement and its variant for online setting. *IEEE Transactions on Knowledge and Data Engineering*, 28(9):2281–2295.

She, J., Tong, Y., Chen, L., and Song, T. (2017). Feedback-aware social event-participant arrangement. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 851–865.

Sinnen, O. and Sousa, L. A. (2005). Communication contention in task scheduling. *IEEE Transactions on parallel and distributed systems*, 16(6):503–515.

Sozio, M. and Gionis, A. (2010). The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 939–948.

Su, X. and Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009.

Su, Y., Wang, B., Cheng, F., Zhang, L., Zhang, X., and Pan, L. (2017). An algorithm based on positive and negative links for community detection in signed networks. *Scientific reports*, 7(1):1–12.

Sun, R., Wu, Y., Chen, C., Wang, X., Zhang, W., and Lin, X. (2022). Maximal balanced signed biclique enumeration in signed bipartite graphs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 1887–1899. IEEE.

Suratanee, A., Schaefer, M. H., Betts, M. J., Soons, Z., Mannsperger, H., Harder, N., Oswald, M., Gipp, M., Ramminger, E., Marcus, G., et al. (2014). Characterizing protein interactions employing a genome-wide sirna cellular phenotyping screen. *PLoS Comput Biol*, 10(9):e1003814.

Tang, J., Aggarwal, C., and Liu, H. (2016a). Recommendations in signed social networks. In *Proceedings of the 25th International Conference on World Wide Web*, pages 31–40.

Tang, J., Chang, Y., Aggarwal, C., and Liu, H. (2016b). A survey of signed network mining in social media. *ACM Computing Surveys (CSUR)*, 49(3):1–37.

Tomita, E. (2017). Efficient algorithms for finding maximum and maximal cliques and their applications. In *International Workshop on Algorithms and Computation*, pages 3–15. Springer.

Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., and Wakatsuki, M. (2010). A simple and faster branch-and-bound algorithm for finding a maximum clique. In *International Workshop on Algorithms and Computation*, pages 191–203. Springer.

Tomita, E., Tanaka, A., and Takahashi, H. (2006). The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical computer science*, 363(1):28–42.

Tseng, T., Dhulipala, L., and Shun, J. (2021). Parallel index-based structural graph clustering and its approximation. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1851–1864.

Uno, T., Kiyomi, M., Arimura, H., et al. (2004). Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Fimi*, volume 126.

Vinayagam, A., Zirin, J., Roesel, C., Hu, Y., Yilmazel, B., Samsonova, A. A., Neumüller, R. A., Mohr, S. E., and Perrimon, N. (2014). Integrating protein-protein interaction networks with phenotypes reveals signs of interactions. *Nature methods*, 11(1):94–99.

Wang, J., De Vries, A. P., and Reinders, M. J. (2006). Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 501–508.

Wang, X. and Liu, J. (2018). A comparative study of the measures for evaluating community structure in bipartite networks. *Information Sciences*, 448:249–262.

Woeginger, G. J. (2003). Exact algorithms for np-hard problems: A survey. In *Combinatorial optimization—eureka, you shrink!*, pages 185–207. Springer.

Xiao, H., Ordozgoiti, B., and Gionis, A. (2020). Searching for polarization in signed graphs: a local spectral approach. In *Proceedings of The Web Conference 2020*, pages 362–372.

Yao, K. and Chang, L. (2021). Efficient size-bounded community search over large networks. *Proceedings of the VLDB Endowment*, 14(8):1441–1453.

Yao, K., Chang, L., and Qin, L. (2022a). Computing maximum structural balanced cliques in signed graphs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE.

Yao, K., Chang, L., and Yu, J. X. (2022b). Identifying similar-bicliques in bipartite graphs. *Proceedings of the VLDB Endowment*, 15(11):3085–3097.

Ye, J., Cheng, H., Zhu, Z., and Chen, M. (2013). Predicting positive and negative links in signed social networks by transfer learning. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1477–1488.

Yim, S., Yu, H., Jang, D., and Lee, D. (2018). Annotating activation/inhibition relationships to protein-protein interactions using gene ontology relations. *BMC systems biology*, 12(1):9.

Yu, K., Long, C., Deepak, P., and Chakraborty, T. (2021). On efficient large maximal biplex discovery. *IEEE Transactions on Knowledge and Data Engineering*.

Yuan, L., Qin, L., Zhang, W., Chang, L., and Yang, J. (2017). Index-based densest clique percolation community search in networks. *IEEE Transactions on Knowledge and Data Engineering*, 30(5):922–935.

Zaki, M. J. and Hsiao, C.-J. (2002). Charm: An efficient algorithm for closed itemset mining. In *Proceedings of the 2002 SIAM international conference on data mining*, pages 457–473. SIAM.

Zhang, Y., Phillips, C. A., Rogers, G. L., Baker, E. J., Chesler, E. J., and Langston, M. A. (2014). On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC bioinformatics*, 15(1):1–18.

Zhang, Z.-Y. and Ahn, Y.-Y. (2015). Community detection in bipartite networks using weighted symmetric binary matrix factorization. *International Journal of Modern Physics C*, 26(09):1550096.

Zhou, R., Liu, C., Yu, J. X., Liang, W., Chen, B., and Li, J. (2012). Finding maximal k-edge-connected subgraphs from a large graph. In *Proceedings of the 15th international conference on extending database technology*, pages 480–491.

Zhou, Y., Rossi, A., and Hao, J.-K. (2018). Towards effective exact methods for the maximum balanced biclique problem in bipartite graphs. *European Journal of Operational Research*, 269(3):834–843.

Zou, Z. (2016). Bitruss decomposition of bipartite graphs. In *International Conference on Database Systems for Advanced Applications*, pages 218–233. Springer.